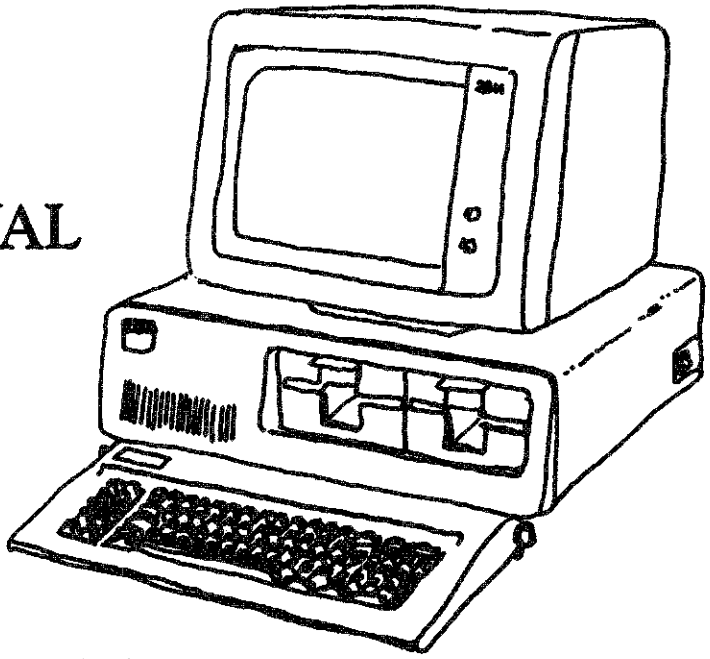
programmer's journal

The Resource Journal for IBM PC Programmers

Volume 1 Number 2 March/April 1983

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROGRAMMER'S JOURNAL



CONTENTS

ASM: WRITING AN ASSEMBLER FOR THE IBM PC / David Whitman 4
How to create your own assembler

IS IT MY HARDWARE OR MY SOFTWARE? / Les Moore 8
Isolate your system problems quickly

PJ PROFILE / Interview 10
Combining the best of software and books

PASCAL & ASM / Ted Lewis 13
Excerpts from his upcoming book

COPY PROTECTION / Tim Anderson 21
The problems of encryption

INTRODUCTION TO USER DEFINED CHARACTER SETS / Mark Minasi 23
Exploring the IBM PC's character sets

C: ACCESSING THE BIOS FUNCTIONS FROM C86 / James E. Randall 27
Using the C language for displaying graphics

FREEWARE 37

DEPARTMENTS

HOUSEKEEPING 2
LETTERS 3
PJ FORUM 15
TOOLS 17
UTILITIES 26
dBASE II 30
PJ LIBRARY 33
SOFTWARE HOUSES 34
MISC. 35

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSEKEEPING

Spelling errors? Aberrations in grammar and syntax? all I can think of is the scene from the movie *Young Frankenstein* when Dr. Frankenstein Jr. turns to his bent sidekick and says, "Wasn't your hump on the other side a minute ago?" . . . to which his sidekick replies, "What hump?"

OK. So we will submit the poetic prose of PJ to spelling checkers and other mechanical devices. You should be aware of the fact that the volume of mail we received concerning our minor "over sights" only encourages us to do it again. The mailbox OVERFLOWED. Actually this is all part of a larger software development plan. It is a secret. We would like to talk about it but can only do so with other programmers bound by the ubiquitous gag order. If you continue to read this page then you are bound in full by a non-disclosure, non-proliferation software agreement.

Whew. Glad you all took the oath. Now we can reveal more of the real story. Sometimes we get a few VIC programmers disguising themselves as professional consultants wearing wrinkled dress shirts and corduroy Levis. At PJ we are working on a new SOFTWARE PACKAGE to end all software packages. We call it our CRC compiler. It will make 1 2 3 look like a boot-strap loader for a hand calculator. The abbreviation stands for "Computer Rag Compiler." There is a tremendous need for copy editing computer magazines and such a dearth of qualified people. So we are implementing a 3 pass compiler for computer magazines. The first pass checks all of the spelling. The second pass adjusts the syntax and grammar. The third pass COMPILES all of the programs in every article. Pretty neat, huh. Sure it will be a little tricky writing a compiler for every known programming language. Maybe it would be good to divide it into 8 and 16 bit CRC compilers. We are open to suggestions. With (not quite) one mega-byte of memory to pump assembly code into there should be room to spare. Perhaps burning it into some PROMs would be better. The best part of the whole project is that we can START ADVERTISING IT NOW. Delivery will not be until Nov-Dec of 1985, but that doesn't matter.

The programmers who wrote articles for this issue deserve special thanks. David Whitman, Mark Minasi, James Randall, Les Moore, Tim Anderson, and Dion Johnson. Ted Rolle helped us with the copy editing. Ted Lewis sent us some source code from his upcoming book. Only when our avocation merges with our vocation is the deed ever really done.

—Greg Estes

PROGRAMMER'S JOURNAL

Editor/Publisher: Greg Estes

Associate Editor: Frankie Detesto

Contributing Editor: Greg Roberts

Art Director: Julia King

Production: Cassandra Estes

Typesetting: Editing & Design Services, Inc.

Writers:
David Whitman
Mark Minasi
Les Moore
Tim S. Anderson
James E. Randall
Ted M. Rolle
Dion Johnson

Editorial and Business Office:
2765 Potter St.
Eugene, OR 97405

Mailing Address: P.O. Box 5070, Eugene, OR 97405

Programmer's Journal. Volume 1, Number 2. Copyright (C)
1983 by Programmer's Journal. Published bi-monthly.

Postmaster: send address changes to Programmer's Journal,
2765 Potter St. Eugene, OR 97405 ph. (503) 687-0398

Subscriptions: Bi-monthly rates are $24 U.S.
($30 in Canada, and $40 elsewhere.)

IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York
dBASE II is a trademark of Ashton-Tate.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

PETER NORTON ARTICLE:

In the procedure 'display—output—string' the column count for the screen-pointer array is not incremented. This results in all the characters in the string being output in the same column on the screen. I believe that for this to work correctly you must replace 'column' with '(column = I + 1)' as the column index into the screen-pointer array.

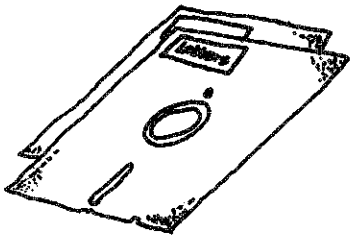
By the way, I purchased a copy of the Norton Utilities last month and think that they are super. There certainly is a need for tools as well done as these. I especially like the screen oriented approach of DISKLOOK.

I work for Intel on the team that designed the iRMX 86 and iRMX 286 operating systems and two of the tools that I can 't live without are the SUBMIT and an enhanced DIR command.

SUBMIT works like a .BAT file except that the batch file name can be anything and invocations of SUBMIT can be nested to any level. This feature is valuable whenever generating a piece of software that consists of more than a couple of files.

DIR should be enhanced with switches to produce multi-column displays and output to a file. Once DIR can output to a file, you can use it to help build SUBMIT files.

Ted Fergeron
Beaverton, OR



PETER NORTON:

While finishing off work on my book ("Inside the IBM Personal Computer" , published by R. J. Brady Co.), I wrote a series of programs to make it easier to produce a book index. The programs run on an IBM/PC with PC-DOS. If you would like a free copy just send me a formatted diskette and a self-addressed return mailing envelope. Be sure and mention the "Book Index Programs" .

Peter Norton
The Norton Utilities
2210 Wilshire Blvd., #186
Santa Monica, CA 90403

DEBUG PROBLEMS:

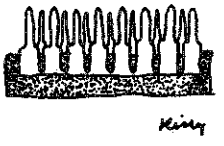
Here is a little tidbit I have discovered. It pertains to why DEBUG can hang up during single step. The condition also occurs while trying to debug a timer interrupt continuator.

1. The 8088 microprocessor that the PC uses has an internal flag, called TF. When this flag has been set by a program (DEBUG), an internal interrupt (TYPE 1) is generated after each of the computer 's instructions is executed. The interrupt forces the computer to a particular memory location, where in the case of the PC, the registers and the instruction just executed are displayed on the CRT.
2. Now, if an external interrupt to the 8088 (TIMER) occurs during the execution of the instruction mentioned above, the 8088 will first recognize the external interrupt. The program counter and flags are pushed to the stack, the TF flag cleared, and the first instruction of the Timer interrupt service routine is executed. After completion of the instruction, the 8088 acknowledges the internal Single Step interrupt by again pushing the flags and the program counter to the stack and starts executing in the DEBUG routine. Here the last instruction address executed is popped from the stack (the address of the Timer interrupt service routine) and is displayed along with the registers and the disassembled instruction.
3. Every thing up to this point is normal for the 8088. It is working as expected. The real problem is, that as far as the PC's interrupt subsystem is concerned (an Intel 8259 Interrupt Controller), the Timer interrupt is the highest priority interrupt pending in the PC. It isn't until the end of the Timer service routine that the interrupt is cleared. DEBUG has no idea that this situation has occurred and goes merrily along its way waiting for a keyboard entry from the operator. The gothca is that the keyboard is an interrupt driven device of lower priority and is locked out by the pending higher priority device, the Timer.

This bug has plagued me from day one. I tried to get it solved through my dealer. All he knew was how to sell the stuff. I wrote a letter to IBM, but they referred me back to the dealer. Apparently they thought I was some type of hobbyist. Well, they still had not fixed it by release 1.1 of the operating system. 1.2, 1.3, 1.4? I have sent the above to the attention of PC magazine. Whether they are interested in such peculiarities is unknown to me.

Professionally yours,

Timothy S. Anderson



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

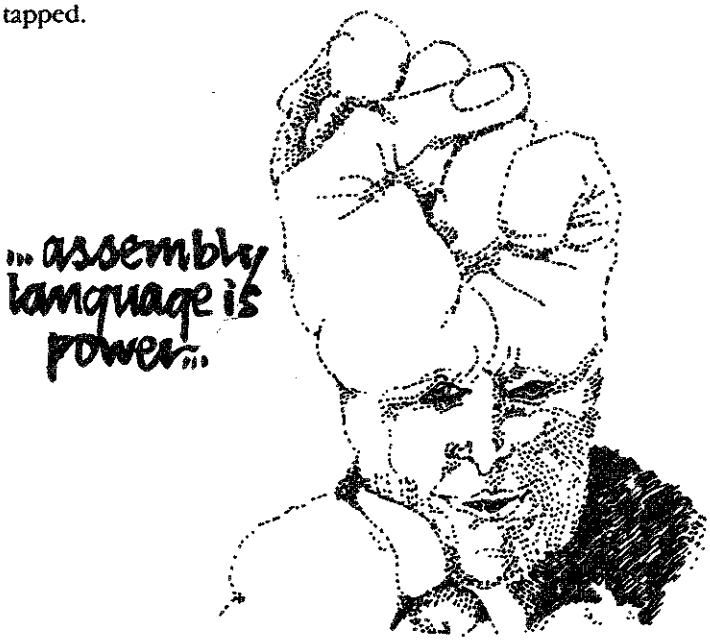
WRITING AN ASSEMBLER FOR THE IBM PC

by David Whitman

Why do people still program in assembly language? Programs in assembler are hard to understand, difficult to debug, and take much more effort to write than equivalent programs in a high level language such as BASIC or Pascal. Why go through all that grief? There are a couple of answers.

Well written assembly language programs run faster than any high level language, even compiled languages. For programs where speed is of the essence, such as game animation, or systems programming, writing in assembler lets you squeeze the absolute top speed out of the computer.

The main reason for writing in assembly language is power. High level languages shield you from the hardware you're programming. This lets you concentrate on the problem you want to solve, rather than on the idiosyncrasies of your computer. However, this shielding process also places limits on what you can do. Assembly language, although complicated, places no restrictions on your capabilities. In addition, it is only through assembly language that many of the powerful services provided by the operating system can be tapped.



For the above reasons, I was interested in experimenting in programming my IBM in assembly language. I rejected the IBM assembler because my PC had only 64K of memory. Although a restricted subset of IBM's assembler can run in 64K, I had read that it was such a tight fit that only ridiculously small programs could be assembled. I couldn't see shelling out $100 for a program which really couldn't run on my machine.

It seemed to me that if computers which are inherently limited to 64K, such as the Apple and TRS-80, could have assemblers, then so could my 64K IBM. Unfortunately, no one seemed to be marketing a product to compete with IBM's offering. That gap was the incentive for writing CHASM (my "CHeap ASseMbler").

THE ASSEMBLY PROCESS:

What CHASM does is to take a file of assembly language statements, and translate them into the corresponding instruction codes for the 8088 microprocessor. Thus, when CHASM reads the line:

```
MOV AX,CX ;move contents of CX register into AX
```

What comes out the other end is the series of numbers:

247 225 (or in hexadecimal, F7 E1)

These numbers are the machine language, or object code, corresponding to the instruction MOV AX,CX. Each line of assembly language corresponds to a single machine language instruction.

Each input line to CHASM may also contain a label. All branching in assembly language is to labels, unlike the line numbers of BASIC. In this respect, assembly language is more sophisticated than BASIC. Compare two subroutine calls:

```
CALL INITIALIZATION—CODE
```

and:

```
220 GOSUB 10000
```

By using a descriptive label for the subroutine, the assembly language call is self-documenting.

In addition to the standard 8088 instructions, CHASM recognizes a set of "pseudo-operations" (IBM calls these "assembler directives"). Some of these pseudo-ops give CHASM special directions about how to assemble the program. For example, the EQU pseudo-op tells CHASM to replace all occurrences of a name with a number. This allows you to give a symbolic name to a frequently used numeric constant, to make your program more readable. Thus, you can write:

```
LENGTH—OF—FIELD EQU 11
        ...
        MOV  CX,LENGTH—OF—FIELD
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Other pseudo-ops allow you to set aside and/or initialize memory space for use by your program. If a memory allocating pseudo-op bears a label, your program can refer to the memory location by that label, rather than by an address, again making for a more readable program. For example, using the pseudo-op DB (Declare Bytes) you could declare the text of an error message, and refer to that spot as ERR—MSG:

```
ERR—MSG DB 'File not Found!'
```

In addition to a file of machine language, CHASM also produces a listing, showing each input line, with the object code produced. If any of the input lines are not grammatically correct, error and diagnostic messages will also appear on the listing, to pinpoint the problem and perhaps suggest a solution.

CHASM does most of the work of translation by table lookup. Input lines are parsed into operation and operand fields. The operands are classified as one or more of the various allowed operand types (for example: registers, immediate data, etc.), then a table of all the allowed instructions is searched for a match of an operation with the given operand types. If a match is found, the table gives the first byte of the object code for the instruction, plus a series of flags which instruct CHASM as to whether or not any further code need be produced to complete the translation. Depending on the instruction, between zero and four more bytes might be necessary to specify the operands.

CHASM is a two pass assembler, which means that it actually does the translation process twice. The first pass does not produce any object code, it merely keeps track of how much code would be generated if the translation process were actually completed. A "location counter" is used to keep a running total of how much code will be produced. During the first pass, the location counter is used to figure out the positions of all the labels in the program, so that instructions to jump to a label, or references to labeled memory locations, can be translated on the second pass. The names of all the labels, along with their offsets, are recorded in a "symbol table" , which is consulted on the second pass every time CHASM encounters a label as an operand. As an aid to debugging, the contents of the symbol table are printed out at the end of the listing.

WRITING CHASM:

CHASM was written in interpreted BASIC, because that was the only language I had access to. This led indirectly to a constant moral battle throughout the actual writing. I knew I was going to have trouble fitting CHASM into a 64K machine, since it was going to be big. On the other hand, the large size of the program made it particularly crucial to liberally sprinkle remarks everywhere, so I could remember how everything worked. Of course, each comment ate up some of my precious memory space. Users who have early versions of CHASM can see a relic of my memory space anxiety; some mysterious unlabeled numbers appear on the screen, which represent the available memory during various stages of initialization (the numbers were ominously small). In later versions, the lines which printed this information were deleted to make more room!

To make more work space, much of the initialization code is gathered into a "transient section" , which is used once, then discarded. This resulted in a savings of about 5K of memory. I got this idea from the DOS manual; on page B-2 the manual mentions that DOS overwrites its own initialization code after it's been used.

I had to resort to a trick in order to erase code from within the running program. BASIC does offer the DELETE command, but this command stops program execution. The trick is to use the delete option of the CHAIN MERGE command. To avoid a syntax error, you have to provide a file to be merged in. Unfortunately, BASIC doesn't provide the dummy filespec "NUL:" which is offered by DOS, but an empty file works just as well. To generate an empty file on a disk, type NEW in BASIC, then SAVE" dummy.bas" ,a. Once the dummy file is in place, you can delete lines (in this example, lines 1000-2000) with the statement:

```
100 CHAIN MERGE "dummy.bas" , ALL, 110, delete ➡
        1000-2000
110 REM This line should be executed next
```

Note the parameter which specifies the line with which to start execution in the merged program. In our dummy merge we want to just continue with the next statement, but we must specify this or BASIC will automatically go to the beginning of the program.

CHASM was written in a top down, "divide and conquer" style. Every time I encountered a problem which seemed too hard to handle, I broke it down into a number of smaller problems. Each of these problems was similarly subdivided until they presented a relatively trivial programming task. Although CHASM occupies about 35,000 bytes of memory, most of the routines are short enough to fit on a 24 line CRT screen. I find the prospect of writing a series of many 20 line subroutines much less intimidating than writing a 1000 line program.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

An added benefit of this technique is that the resulting subroutines tend to be quite modular. If you suspect a problem with a given routine you can unplug it from the rest of the program and test it in isolation. Similarly, if you want to improve some aspect of the program, this usually involves modifying only a small subroutine, rather than the entire program.

The "divide and conquer" strategy applies to other languages besides BASIC. Programmers writing in any language may find their programs more manageable using this technique.

GETTING STARTED:

I am in the process of learning to program in assembly language. Writing CHASM was a major milestone along that path. I'll close out this article with some suggestions to others who are considering trying out this area of programming.

First off, do some reading. Get a book which describes the the 8088 instruction set, and explains the various addressing modes. I have read "The 8086 Book" (1980, Osbourne/McGraw-Hill) and found it to be quite useful, if somewhat terse. It makes an excellent reference book. A number of other books about the 8086/8088 family exist. Since I haven't yet read them, I'm unable to review them.

Also required reading are the DOS manual and Technical Reference. The operating system and BIOS provide simple access to such functions as screen i/o and file management. To the assembly language programmer, these services represent high level commands added to the basic instruction set of the 8088. Although it would be possible to write code to perform these functions, that would be re-inventing the wheel. Become familiar with what the operating system can do for you.

What the above two steps give you is an idea of what basic moves are available in assembly language. What it probably doesn't do is teach you how to bring it all together into a working program. My suggestion here is to learn by example, from the experts. Using the unassemble command in DEBUG, you can convert machine language programs back into something very close to assembly language. I've picked up a lot of tricks by unassembling some of the utilities on the DOS diskette. Peter Norton has written a very nice article in the November 1982 issue of Softalk magazine about unassembling the BASIC interpreter. I'd recommend digging up that article and reading it. Practice using DEBUG; it is a very powerful tool. Besides allowing you to look at what other people have written, you'll be testing your own programs in this environment. A little time spent familiarizing yourself with the various commands will be repaid many times over when you try to kill the bugs in your own work.

At this point, it's time to start writing your own programs, and you'll need an assembler. If you have lots of memory, and $100, the IBM Macro Assembler is available. Although I do not own a copy, examining the documentation convinced me that the program is worth the money, for someone whose expertise and system resources allow them to exploit its full power. If you're short on money, memory, or experience, I recommend that you get a copy of CHASM, either from me, or from a friend who already has it. CHASM is distributed free of charge, the only catch being is that I request a $20 donation from those who find the program useful.

Your first programs should be very simple, perhaps calling a single BIOS routine to see what it can do. Try printing messages on the CRT using the DOS string printing function. Don't try anything fancy at first, because you're learning a totally new type of programming.

In particular, avoid conditional branching at first. In machine language, all branching decisions are based on the state of the flag register, which is set after executing each instruction. This convention, while powerful, is somewhat confusing, and I'd defer struggling with it at first. It's possible to write useful programs without any branching at all. □

The assembler described in this article is available free of charge by sending a disk and a stamped return mailer to the author. Send disks to:

David Whitman
Department of Chemistry
Dartmouth College
Hanover, NH 03755

A $20 donation is requested from all satisfied users.

Listing 1 is an example of a good beginning program. The program is short, with no branching. Two of the features of BIOS function 10, VIDEO—IO, are experimented with: scrolling the video display, and positioning the cursor. By typing the program's name (CLS) from DOS, the screen will be cleared. This adds an external command to DOS, equivalent to BASIC's CLS statement.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

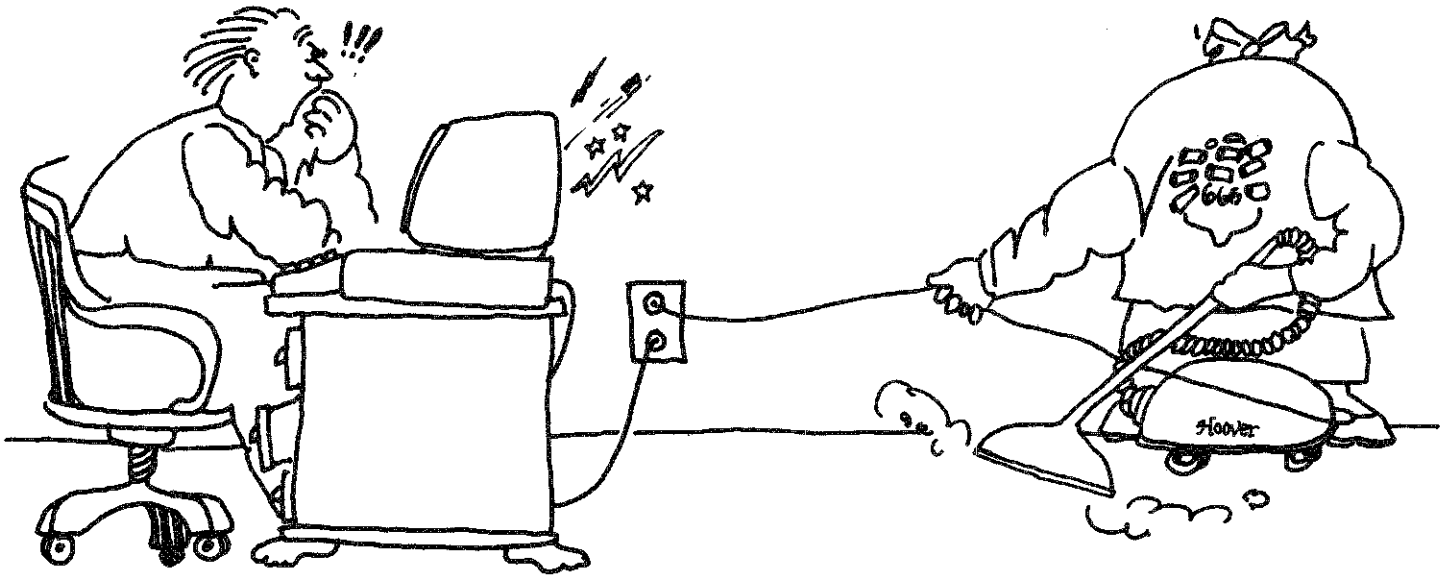
LISTING 1

cls.asm

```
LOC  OBJ            LINE  SOURCE

0100                   1  ;-----------------------------------------------;
0100                   2  ; PROGRAM CLS  version 1.0  by Dave Whitman     ;
0100                   3  ;                                               ;
0100                   4  ; Provides a DOS external command to clear CRT. ;
0100                   5  ;                                               ;
0100                   6  ; Calls BIOS routine VIDEO_IO to scroll the screen ;
0100                   7  ; up, and to position the cursor in the upper left ;
0100                   8  ; hand corner.  See page A-43 of Technical      ;
0100                   9  ; Referance for documentation of VIDEO_IO.      ;
0100                  10  ;-----------------------------------------------;
0100                  11  CLS  PROC FAR
0100                  12  ;
0100 B80006           13       MOV AX,0600H    ;scroll function, entire window
0103 B90000           14       MOV CX,0000H    ;upper left corner (0,0)
0106 BA4F18           15       MOV DX,184FH    ;lower right corner (24,79)
0109 B707             16       MOV BH,07       ;normal attribute
010B CD10             17       INT 10H         ;call VIDEO_IO
010D                  18  ;
010D B402             19       MOV AH,02H      ;select cursor set function
010F BA0000           20       MOV DX,0000H    ;move cursor to (0,0)
0112 B700             21       MOV BH,00H      ;page number
0114 CD10             22       INT 10H         ;call VIDEO_IO
0116                  23  ;
0116 CD20             24       INT 20H         ;return to DOS
0118                  25       ENDP


 0 Error(s) detected
 0 Diagnostic(s) offered

SYMBOL TABLE DUMP:
CLS         0100
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HARDWARE OR SOFTWARE



"Can't hear you honey... I'll be done in just a few minutes!"

IS IT MY SOFTWARE OR MY HARDWARE?

by Les Moore

(All of us will have to face software bugs and hardware problems at some point in our personal computer careers. The bugs may be in our own system, a friends, or a clients. As long as the problem can be readily diagnosed it is normally easily solved. The difficult first task is trying to decide if it is the machine or the programs. Les Moore, experienced in both, shares his insights.)

Let's say there is something wrong with some data. You have found a garbage record. How do you approach figuring out what caused it? Not only do you have to correct the bad data you more importantly have to fix the cause.

Often you have some intuition as to the cause. If you suspect it is the hardware then you should first run a diagnostic test of the memory, as well as running a check on your diskette. Data is stored both in memory and on the diskettes surface. Run a memory test to see if any of your chips are bad. If that looks good then you should check the diskette surface and the disk drive. If you try reading the diskette in the other drive and it works then you know your disk drive is faulty.

Isolating these sorts of problems can often be accomplished by interchanging hardware and software. This quickly narrows down which area it is in. If you are reasonably sure it is the hardware then take the software over to a friends house and see if the identical problems occur. If your software runs smoothly on another machine then you know your hardware must be at fault. But, if you run the software on another machine and the same problems reoccur then you know it is the software. There are of course other possibilities, different versions of the operating system or subroutines,but that is usually not the case.

Look at the pattern of the bad data. If it is localized to one record then it is probably not the diskette surface. But if several records right next to each other are garbage then it looks like the media may be the problem. If only one distinct record is wrong, then it is probably the software or your memory, or maybe one bad rewrite from the head. If all the records have a fairly high sector number or a fairly low sector number then it may be the drive. As the drive gets further and further out of alignment this can happen. You may find you can read most of someone else's diskette except the beginning or ending sectors. Here again it is probably a disk drive alignment problem.

If hardware problems are suspected then a number of environmental situations should be looked at. There are 2 extremes of temperature to be aware of for your computer, very hot or too cold. If you transport your computer in a vehicle that is quite cold then it may take an hour for the machine to warm up before it functions correctly. Normally cold temperatures are not the cause of computer malfunctions. Whenever my computer room exceeds 80 degrees it starts to bother me.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Parity errors are normally caused by hardware malfunction but there are known cases where a new piece of equipment or operating system software make it extremely difficult to pin point where the problem is. Parity errors can be caused by many things, like power surges while you are running your machine. Most computers should have a surge suppresser installed along with them.

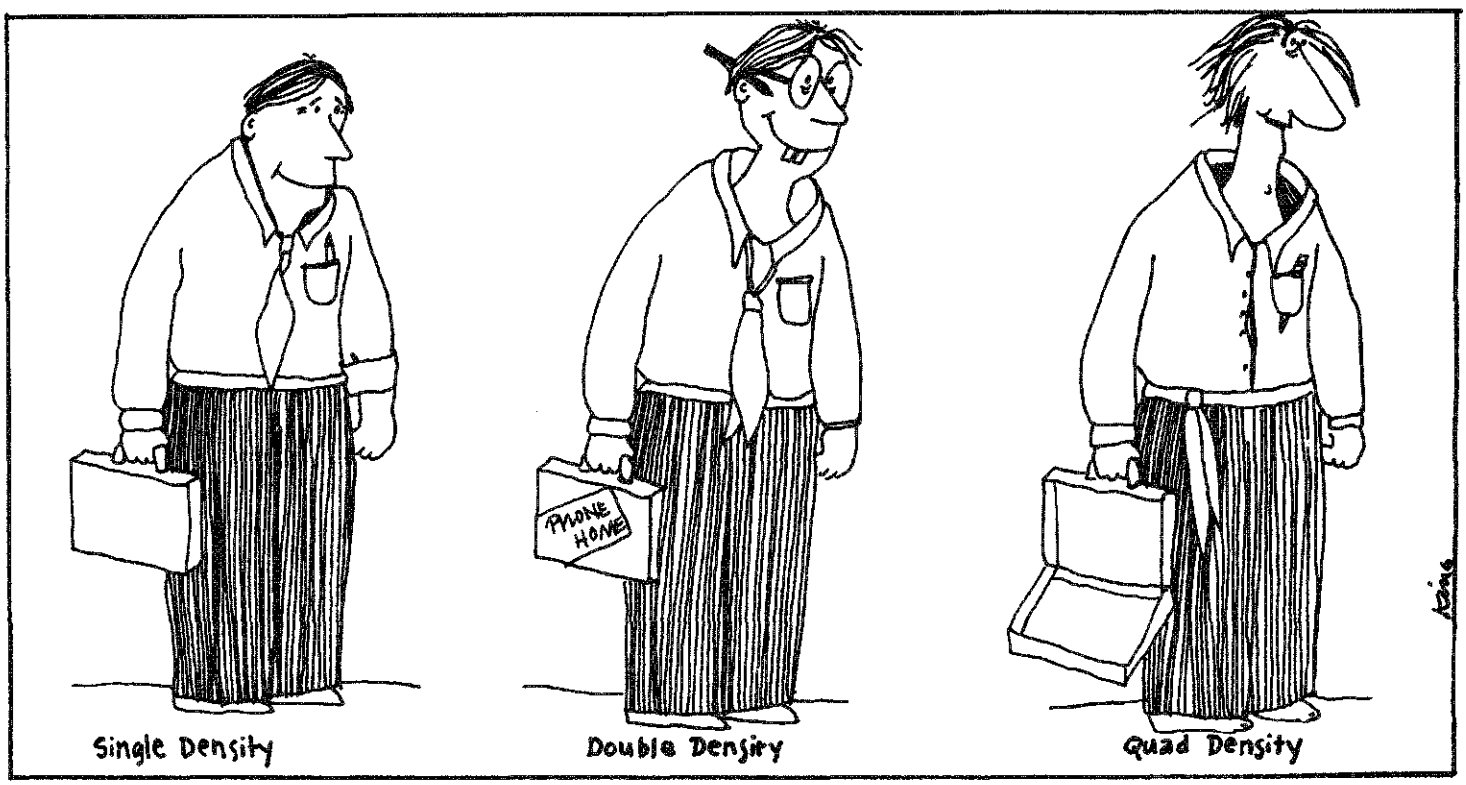
Some users have noticed that their problems occur at a particular time of the day. It may be when air conditioners are switching themselves on, or late in the day when the room may still be under 80 degrees but the sun is hitting the computer directly and the internal temperature has risen too high. Power failures can do quite a lot of damage. We forget how often temporary breaks in home current happen. If a major drop in power occurs and you are using a hard disk it could cause the heads to drop directly onto the platters. I remember setting up a Cromemco system at our local tennis club. After they had entered a great deal of information the power went out and the drive heads dropped onto the surface of the disk. I spent hours trying to manually link the blocks back together.

I find myself avoiding certain situations that seem to increase the chance of computer errors. I try to remove the cover as little as possible and restrict plugging in and pulling out of the boards. The contact between the board and the socket can be thrown out of alignment. If a cross contact is made then various chips will be burnt out. I once watched an extremely experienced repair technician just push in a board with the power on and destroy quite a few chips. Another example would be if you put your computer in the trunk of your car and didn't have it absolutely level. You would be putting a lot of torque on the chassis which could disrupt all kinds of contacts. If I setup my computer after a trip and it fails to work, I take the top off and remove all the boards and carefully reseat them one by one. All in all I have had little trouble with a poorly seated board damaging itself. Usually it just fails to work properly.

Ideally you should have a separate circuit for your computer. No air-conditioners or refrigerators connected to the same line. Really anything that has an AC motor that turns on and off. If someone is running a vacuum cleaner or a power drill while you are computing you can run into trouble. I saw a TI-810 printer totally destroyed once when someone used an improperly wired vacuum cleaner on the same outlet. It just burnt up.

If you install your own computer chips you need to limit the static electricity. Do not install them if you are working on a carpeted floor and be sure and discharge yourself on something metal before beginning the project. Chips are becoming better and better so this is not an enormous problem. Damaged chips are usually a result of putting them in backwards, with the notch on the wrong end or bending one of the pins while pushing them in.

If you keep tabs on your computer's environment you can avoid many of the common system malfunctions. You should mentally take note of any major changes you make and see if your computer is affected as well. □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ PROFILE

an interview with Ted Lewis

(Ted Lewis is the author of Data Design, a database manager for the IBM PC marketed by Insoft, as well as numerous computer books.)

PJ: When did you buy your IBM PC.

TED: Well, when IBM announced their personal computer I immediately ordered one but had to go through the IBM Product Center in San Francisco because locally there was such a waiting list. I knew that it would profoundly affect the computer industry regardless of what the actual machine could do. I was surprised that it was as well done as it turned out. You can tell they did their homework.

PJ: Have you owned any other computers?

TED: The first one I bought was an IMSAI with NorthStar drives and NorthStar BASIC. That was in Eugene Oregon at the Real Oregon Computer Store around 75 or 76. I began running UCSD Pascal as soon as it was out. Then I bought an Apple and wrote a book on Pascal programming for it. I also had an Atari but never produced anything for general consumption. Then onto an Osborne and finally the IBM PC.

PJ: What is your computer background.

TED: I have Masters and Ph.D in computer science and a B.S. in Mathematics. I think this year makes 20 years that I have been working with computers. I started with vacuum tube machines back in 1963 at OSU (Oregon State University.). I worked as an operator for a while. The first language I learned was SmalGol, it was an ALGOL derivative. In one sense I have been a Pascal programmer since the inception of that language. The second language I learned was FORTRAN. Then I worked for Boeing as a programmer, then Sylvania for 1½ years in Sunnyvale, California. I kept having to fund my own schooling so I would work for a while then go back for more education. It took me 11 years to complete my Ph.D in 1971 which marked the end of my formal training. This was really the beginning of computing because that was the year the microprocessor was invented. The entry of the 8008 and so on. To me everything that happened before 71 doesn't count a whole lot.

Historically computer centers have been big empires. They were very powerful entities in an organization which required you to go to them to use the machine and access the information you wanted. The advent of personal computers just devastated the computer centers because they could no longer maintain control. It is the same whether it is in a University or a Fortune 500 company. Many departments within an organization had to purchase their personal computers under such names as "data collecters" , "controllers" or "loggers" when in fact they were Apples or IBM's. It was the simplest way to avoid the careful scrutiny they always encountered whenever they needed to purchase a "computer."

PJ: How would you describe your working style as a programmer?

TED: I tend to function in a burst mode, accomplishing a lot over a short period of time and then take a break. I would put programmers into 2 groups at least. Group 1 works well in large organizations and is compatible with top down design, and then there are the renegade programmers who work best as bottom up programmers. The bottom up person is more of a ad hoc programmer. He prefers writing all of the low level routines, likes assembly language and generates the application package from the small routine to the large. Top down style implies a larger organization with more external controls, more documentation, and the marketing/design/modular structure. Everything from the screens to the individual modules are completely laid out. It is difficult to work in a top down environment if you are a bottom up type of person.

PJ: Have you found any special difficulties writing software?

TED: The software author is always faced with a changing dilemma. What machine to use, what language to use, which operating system. All of which is extremely critical. Look at all of those firms that did not go with CP/M. Vector Graphics, Altos, and Osborne all chose CP/M which at that time may have been merely coincidental. Look at the companies that chose other alternatives. North Star stuck with it's own operating system, then there is Oasis. Apple was helped tremendously by VisiCalc's success. You can look at the trend lines of Apple sales versus VisiCalc sales. Even though Apple stuck with it's own operating system it still developed a softcard to allow CP/M to run on it. Using

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the p-system has many advantages, one of them being the ease of transportability, however Softech Micro Systems is killing that because of their licensing strategy. You can sell run-time support of the p-system with your application for around $1.50, but to get that kind of pricing you have to buy 40,000 copies and pay the fee in advance. Only a large organization can afford to pay that much in advance, and many of them are also unwilling to pay such advances. That is why their development systems are selling well but the applications produced from them are not frequently marketed.

I am not sure about their most current pricing schedules but last time I checked the minimum number you could purchase was 200 at $35 a copy. This completely precludes any inexpensive, under $50 packages from being marketed. You have to figure that a software developers income is about a 10 to 1 ratio. For every $1 in profit he has to sell a package for $10.

PJ: Say for example you were selling a $50 game through a distributor, what would be the breakdown in the retail dollar?

TED: Well the distributor takes out 60% then the publisher takes out 75% from what he receives from the distributor. The software author receives usually 20 to 25% of what the publisher gets. It generally works out to 10% of the retail price. Any time you are required to pay extra for a run-time support package your profits drop accordingly. There is still some speculation as to which operating system will dominate in the 16 bit world. Certainly the p-system would have a good chance against Unix if they changed their run-time pricing strategy. Right now MS-DOS is taking a lions share because you only have to pay $40 for it. CP/M started the same way with a $40 price tag and took over the micro market. Now Digital Research is charging $250 for CP/M-86 and it is just not selling.

(Since the interview Digital Research has dropped the price of CP/M-86 dramatically. The cost is very close to that of MS-DOS.)

PJ: How vulnerable do you feel software authors are once they have signed a marketing agreement?

TED: I was just reading the article in SOFTALK about John Draper. That sort of thing occurs all to often, certainly more than we would like to imagine. Once a programmer signs a contract he has little control over what follows. How long it takes to get the software to the market, or what type of advertising is used. You have basically signed over the rights to everything.

PJ: Aren't monthly advances a possibility?

TED: Yes, advances are your best defense. Say $500 to $1000 per month for 6 months to a year during development. These are considered advances on future royalties. It is the carrot and the stick approach. They can cut off the payment anytime if they decide to cancel the project but you at least get something for your work.

PJ: How about selling the idea before writing the programs?

TED: Selling an idea depends on your visibility. If you have done previous projects and demonstrate your capabilities you are able to do this better. A publisher is less willing to take a chance with an unknown quantity. Once you are somewhat of a known entity you can begin to make a living off of advances. It is still your own money in the form of advance royalties. The publisher is much more at risk in the long run because of their advertising investment. Consequently they will not usually bring up the subject of advances on their own.

PJ: Do you have to "sell" yourself to the publishers?

TED: Obviously the more versed you are at PR the better your chances will be at negotiating software contracts with advances. I do not consider myself particularly good at marketing myself. There are in fact brokers that help larger firms by supply them with independent programming talent to offset their overabundant workload. They connect the free-lance software author with the appropriate company.

PJ: Do you feel that book authors and software authors are basically in the same boat?

TED: Their problems are very much the same, however software authors suffer an additional hazard. The software distribution channels are very narrow. If you do not make it with SoftSel, Micromedia, or the other large volume distributors there will essentially be no market for you. That is the reason you will be seeing more movement in the non-traditional computer store type outlets (bookstores.) Book publishers are rapidly entering the competition. There are about 3,000 computer stores in the US and about 45,000 book stores.

PJ: How many computer books have you written?

TED: Let's see, 12 have already been published and there are 2 more due to come out this year. The earliest one was published in 1975. I have books that are SO BAD that only 100 people actually own some of them. That was in the early days. Basically one of them was my thesis. It got high praise from the people who cared. It taught me a lesson about writing; that highly technical books do not sell well. There is a place for that.

PJ: How many publishers have you dealt with?

TED: Lexington Books, Hayden, Houghton-Mifflin, Wiley, dilithium Press, Reston, and Addison-Wesley.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ: Did you sell the idea to them first or actually write the book?

TED: Originally in the early days I would have to go around and peddle my wares. Now I usually get calls from publishers asking if I am interested in doing a particular piece. It can get to the point where a publisher will send you a blank contract, they don't care what you do as long as you do it with them. So it is not hard to get work once you are a known quantity. This has only been a recent occurrence with me. It is a result of the computer industry really beginning to roll.

PJ: Do they normally give you advances?

TED: Yes.

PJ: Do you normally have a deadline?

TED: Usually. They don't have trouble with me finishing a book. The publishers nightmare is the half completed book. They end up paying an advance and only have half a book to show for it. If I can't complete a book in 3 to 6 months then I can't make any money doing it. I like to do anywhere from 2 to 4 books a year. Last year I did 6 books of varying quality and depth. But that is too many.

PJ: What is the probable life span of a computer book?

TED: About 3 years. If you do a text book you can stretch that figure a little.

PJ: Do you determine the length of your books?

TED: Generally, but the publishers try and control that too. You see there is a psychology surrounding book design. A paperback book that is not too large and feels good in your hand has an advantage over a thick hardback book. So there are definite reasons not to do big books. The largest one I've done was an advanced software engineering book, hard back, 500 pages and sells well in the textbook market. You tend to want physical substance with a textbook. Now if you tried to do that with a personal computer book it would be too intimidating. The buyer would be afraid that they would have to learn too much. Take one of the best selling computer books in the world "Computers for Everybody." It is the 8th best seller for the Dalton chain, it does even better than their sex novels. It will probably sell over a quarter of a million copies this year. It has an attractive cover, it is not big, the lettering is large, with lots of pictures, unintimidating pictures and text, and finally it is informative at a very light level. You can tell your reader to use an RS-232 interface to attach a printer or you can just say you can plug it into a printer. People that buy books by the thousands do not seem to want the heavy book, jammed with facts from cover to cover. On the other hand I have seen fellow University instructors who were in the process of picking out textbooks take one look at a glossy non-technical photo and then chuck it out.

PJ: Are your books more technical or tutorial in nature now?

TED: The first ones were very technical and esoteric while all of my current books use an everyday street language approach. This came about as a reflection of the course the computer industry has taken. If I took a book I've written this year and tried to market it 5 years ago it wouldn't sell at all, and the converse is true. The books I've done have unconsciously paralleled the evolution of computer literacy. First there were hobbyists, then programmers, then application writers, and now users. The book writers and programmers that have been able to adapt to this changing emphasis are doing quite well today. If Adam Osborne tried to produce Volume 1 of his computer books in 1983 very few people would buy it. Back then you needed to build your own computer, now it is all done for you.

PJ: How do you handle the situation of being asked what would be a good computer to buy?

TED: A fellow came to me a while ago from the Math Department very worried about which computer to buy. He was concerned about making a big, expensive mistake. I said look, they are all throw-away computers. You buy one, you use it then find out you have out grown it, then you sell or discard it. The time you spend trying to figure out which one to use you could be spending learning. The education alone is worth the money. He purchased several computers and then a year later decided they should have a different kind. He said he didn't regret his first choice. In one sense it is not the right approach to ask a computer professional which home computer to buy. A novice should ask their friend the pizza store owner what they bought and liked. Computer experts always have to keep biting the bullet in order to stay up with the cutting edge of technology. We all end up with a few arrows in our backs as a result. There are certainly a fair number of obsolete computers sitting around in our back rooms attesting to that fact. We just hope we don't bite too many bullets in the same year. I suppose the advice we can give would be to tell someone what is "safe" to buy. We can give them a list of computer brands that they can't go to far wrong on.

PJ: What about the future?

TED: The Data Design database manager will get a few enhancements. This may be done by other programmers who will in turn receive a share of the royalties. This is a "federation" approach to developing software. In a large

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

organization with a great deal of overhead the software author codes smaller pieces of a larger product and is paid a smaller salary. There is no reason why several programmers cannot join forces and form a software development group. This would allow them to reap the full financial benefits of their work rather than lose most of it in overhead. The time to market problem is also greatly reduced with 2 or 3 programmers working on the same project. It is a bottom up method of accomplishing major software packages. Writing quality software may or may not make you a lot of money depending on the channel you use to market it. There is no reason why several programmers located in different places in the US cannot collaborate on the same project. One programmer is the "software editor" or spokesman for the rest. Probably the most gratifying aspects are the high motivation for completing the project quickly coupled with the substantial monetary reward resulting from no corporate overhead. The rewards of free-lance programming are multiplied in this "guild" form of partnership.

SAMPLE IBM PASCAL PROGRAMS FOR THE IBM PC

The following programs show how to implement the PAGE and GOTOXY routines found in UCSD Pascal in the MS-DOS IBM Pascal Language. Included are both the Pascal code and the assembler routines needed.

—program TESTUNIT (stored in MAIN.PAS) shows how to use the unit containing TAB, PAGE, GOTOXY, and XYPOS for the monochrome adapter (it should work for the color adapter, too).

—program SCREENSTUFF (stored in UNIT.PAS and SCREEN.STF) shows how the TAB, PAGE, GOTOXY, and XYPOS intrinsics are implemented. They should be compiled and put in a link module called SCREENIO, et al.

The assembler level routines needed to access MS-DOS entry points (via interrupts or traps to DOS) are shown in the *.ASM files. They are LOCATE, CURSE, and CLEARSCR. They also show how to interface to Pascal conventions. These routines must be assembled and put in an appropriate link module, too.

(These programs will be published in a forthcoming book by Addison-Wesley. They are reprinted here courtesy of Addison-Wesley and all rights are reserved).

```
{$include:'screen.stf'}
program TESTUNIT( input, output );

 uses SCREENSTUFF;

 var
  ROW, COL, I : integer ;
  OK : char ;

begin
 repeat
  PAGE ;          { clear the screen }
  write( TAB( 30 ),'Enter row, col =') ; readln( ROW, COL );
  GOTOXY( ROW, COL );    { move cursor to ROW, COL }
  XYPOS( ROW, COL ) ;    { get ROW, COL back from screen }
  write( '.(', ROW:2, COL:2, ')' ); { display a dot }
  readln( OK );      { press any key to continue }
 until false;      { never ends -- use CTRL+BREAK }
end.     { TESTUNIT }
```

```
 {$include:'screen.stf'}

 implementation of screenstuff;

  procedure locate( var POS : word ) ; extern ;
  procedure curse( var POS : word ) ; extern ;
  procedure clearscr ; extern ;

  function TAB ;
   var I : integer ;
   begin
    for I := 1 to COL do write( ' ' ) ;
    TAB := chr( 0 )
   end; { TAB }

  procedure PAGE ;
   var BROW, BCOL : byte;
       POSXY : word ;
   begin
    clearscr;
    BROW := 0; BCOL := 0; POSXY := byword( BROW, BCOL );
    locate( POSXY )
   end; { PAGE }
```

```
interface(1);
unit screenstuff( TAB, PAGE, GOTOXY, XYPOS );

 function TAB( COL : integer ) : char;

 procedure PAGE;

 procedure GOTOXY( ROW, COL : integer );

 procedure XYPOS( var ROW, COL : integer ) ;

begin
end;
{ end of screenstuff interface part }
```

```
procedure GOTOXY;
 var BROW, BCOL : byte ;
     POSXY : word ;
 begin
  if ( ROW > 0 ) and ( ROW <= 24 ) then
     if ( COL > 0 ) and ( COL <= 80 ) then
      begin
       BROW := lobyte( ROW - 1 );
       BCOL := lobyte( COL - 1 );
       POSXY:= byword( BROW, BCOL );
       locate( POSXY );
      end { if }
 end;  { GOTOXY }

procedure XYPOS;
 var BROW, BCOL : word;
     POSXY : word;
 begin
  curse( POSXY );  { get x,y-coordinates in POSXY }
  BCOL  := lobyte( POSXY );
  BROW  := hibyte( POSXY );
  COL   := ord( BCOL ) + 1 ;
  ROW   := ord( BROW ) + 1 ;
 end;  { XYPOS }

begin
end.  { implementation }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;*********************************************************
;    LOCATE : Move cursor to X- Y- coordinates
;*********************************************************
; Declare this routine from Pascal as follows :
;
; procedure locate ( var pos : word ); EXTERN;
;
; local data structure for register save, etc.
frame   struc       ; local frame structure
savebp  dw      ?   ; save caller's base pointer
saveret dd      ?   ; return address to Pascal
pos     dw      ?   ; move to POS ( 2-byte x- y- coord )
frame   ends
;
cseg    segment   'code'
dgroup  group     data,stack,const,heap,memory
        assume    cs:cseg,ds:dgroup,es:dgroup,ss:dgroup
;
;
locate    proc    far
          public  locate
          push    bp      ; push caller's base pointer
          mov     bp,sp   ; point to local frame structure
          mov     si,[bp].pos ; setup registers for interrupt
          mov     dx,[si]
          mov     bh,0    ; screen number=0
          mov     ah,2    ; command=2 to move cursor
          mov     al,0    ; low byte not used
          int     10h     ; trap to DOS. Let DOS do it.
          pop     bp      ; restore the base pointer
          ret             ; return to Pascal
locate    endp
cseg      ends
          end
```

```
;*********************************************************
; CLEARSCR : Clear the screen ( 0 )
;*********************************************************
; Declare this routine from Pascal as follows :
;
; procedure clearscr ; EXTERN;
;
; local data structure for registers, etc.
frame   struc       ; local frame structure
savebp  dw      ?   ; save caller's base pointer
saveret dd      ?   ; return address to Pascal
frame   ends
;
cseg    segment   'code'
dgroup  group     data,stack,const,heap,memory
        assume    cs:cseg,ds:dgroup,es:dgroup,ss:dgroup
;
clearscr  proc    far
          public  clearscr
          push    bp      ; save caller's base pointer
          mov     bp,sp   ; point to local frame pointer
          mov     cx,0    ; upper left corner
          mov     dx,2479h ; bottom right position
          mov     bh,7    ; normal attribute...don't ask it!
          mov     ax,600h  ; clear entire screen
          int     10h     ; trap to DOS. Let DOS do it.
          mov     dx,0    ; ok... put cursor at 0,0
          mov     bh,0    ; screen number=0 for monochrome
          mov     ah,2    ; command=2 to move cursor
          int     10h     ; trap again to DOS. Let DOS do it.
          pop     bp      ; restore Pascal base pointer
          ret
clearscr  endp
cseg      ends
          end
```

□

```
;*********************************************************
;  CURSE : Return the cursor position to caller
;*********************************************************
;
; Declare this routine from Pascal as follows :
;
; procedure curse  ( var spot : word ) ; EXTERN;
;
; local data structure for register save, etc.
frame   struc       ; local frame structure
savebp  dw      ?   ; save caller's base pointer
saveret dd      ?   ; return address to Pascal
spot    dw      ?   ; (dh,dl) = row, column
frame   ends
;
cseg    segment   'code'
dgroup  group     data,stack,const,heap,memory
        assume    cs:cseg,ds:dgroup,es:dgroup,ss:dgroup
;
;
curse     proc    far
          public  curse
          push    bp      ; save caller's base pointer
          mov     bp,sp   ; point to local frame pointer
          mov     ah,3    ; command=3 to retrieve position
          mov     al,0    ; low byte not used
          mov     bh,0    ; page number must be 0 for monochrome
          int     10h     ; trap to DOS. Let DOS do it.
          mov     si,[bp].spot  ; fetch the retrieved position
          mov     [si],dx       ; and pass it to Pascal
          pop     bp            ; restore Pascal base pointer
          ret
curse     endp
cseg      ends
          end
```


OPTIMUM-SORT™

SORT/MERGE program for the IBM-PC

More powerful & flexible than programs costing twice as much!

Features

- Sort ascending or descending on up to nine fields
- Five input files may be sorted at one time
- Merge up to five files into one file
- Written in assembly language for high performance
- Dynamically allocates memory and work files
- Filesize limited only by your disk space
- Supports all common data types
- Supports variable and fixed length records
- May be run from keyboard or as a batch command
- Easy to use, includes on-line help feature
- Full documentation — sized to fit your PC manuals
- **$75** — VISA, MasterCard, Check, Money Order or COD

To order or to receive additional information write or call:

OPTIMUM DATA PROCESSING

P.O. Box 2167 Humble, Texas 77347

(713) 454-7428

Requires PC-DOS, 48K and One Disk Drive

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

.cm Sample letterhead to demo IBM 6670 capabilities
.fo off;.hs 0;.tm 0
.sf;.bf op63

JOHNSON AUTOMOTIVE - DIGITAL STUFF DIVISION

I'm addressing a few comments to you about your publication. Free advice is usually worth what you pay for it... and I find that people tend to value things in proportion to what they pay, not what the things are worth. That's life. Nevertheless, your philosophy about passing on that which was once freely given is a nice one, has a certain existential charm to it...

Re Ted Lewis' excerpts: mostly good points, BUT:

Besides the no-assumptions rule, a program should, so far as the facilities permit (in some cases, this requires inventing some facilities), SAVE and RESTORE the state of the machine before and after the program runs. No one wants to find his whatever has been altered after some program runs, unless it was supposed to do that, for example.

To dedicate certain keys to certain functions is only acceptable on machines where the keys are not programmable. ("Go to next page when space bar is pressed"). Fooey. I want to have a profile for each program I use which specifies which keys I want associated with which functions. On my IBM PC, I have keys labelled PgUp and PgDn —these would be the logical scrollers.

Likewise, the scroll amount should be variable - not 1 page or a half page. I want to use a key (or joystick) as a forward/back throttle to control continuous line-by-line scrolling, at a variable rate.

Even nicer than the "please wait I am working" message, is to display something, even anything, while the machine is grinding. To tell the user to wait is an admission of shortcomings. Use a little PR savvy. I know it's gimmicky, but end-users and demonstratees are impressed by flashing gizmos, or whimsical trash. One of my colleagues wrote a big-blue-to-little-blue downloading program which displays the lines of hexadecimal as they come rolling by at 9600 baud, and it's more fun to see that than nothing at all. Mesmerizing, really, or stony.

Another maxim which is not so obvious, but seems to be successful is: If the program can't do what is asked, then stop, explain why and DON'T TRY to fix things or do anything more.

I take issue with Ted's suggestion to use menus wherever possible for command inputs. This falls apart completely when one tries to combine programs and make them procedure-driven. We don't want some program which is n levels down in the calling sequence suddenly presenting the user with a menu of totally irrelevant (to his problem) commands.

I disagree about requiring the RETURN key for all inputs. For really fast inputs, keystrokes need to be reduced. Programmers are used to hitting ENTER - experienced word processors are not so hidebound.

HELP screens are great, and Ted is right. HELP should ALWAYS be available for novice and expert alike. Several levels of error messages are nice, if you can afford it.

Re COBOL: I didn't read it. COBOL sucks, no matter how many governments require it. To all the offended COBOLers: I'm sorry. You probably are too.

Re PJ Forum: Frankie and the wart-hogs (a new-wave band): Frankie should stick to playing the electric interface. "Every language is a trade off." This is either sloppy grammar, fuzzy thinking or a content-free generalization. I'd have liked to know the REAL times for the Pascal comparison. If all things were equal, we would not have to choose. This article must have been written after a long, wet lunch. Didn't he hear what I said above about the programmable keyboard? You can have your keys and change them too.

Greg Estes' article on BASIC tools of the trade is fine - nothing to complain about here.

Re C: A nice little article. I especially like the term 'syntax-ridden'. I've thought that Pascal was hyper-syntactic; was hoping C was not. With IBM of course the parochialism was to never touch any Bell Labs goodies, just as the Bellabers used to choke sooner than use APL. Now that's changing, of course, since the IBM PC needs a decent systems programming language and since IBM has kept their systems languages unavailable, they must face a public that only knows other companies' systems languages.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Re Frankie's article on dBASEII - I liked it. I like enthusiasm. It's much more impressive than a positive but cold professional judgement - no sarcasm implied.

Re System Analysis: I've read so many similar books and diagrams and such crap that this seemed like a rehash-mishmash of them all. I never did understand this stuff, and was relieved to find at the end that it was proprietary information that I had no right to it at all. Thank Dog, I wasn't able to use any of it.

Programming Utilities - Reviews of software are becoming an important class of useful information in the programming business. No one has time to check out all the new software - we must rely on critics to help cull the useful from the whole mess.

Re RANDOM NOTES: The hair-dryer test is appealing - agreed, one doesn't want to perform this on a personally owned system. Those driers heat up circuits way higher than they should. If you didn't have memory problems before, you surashell will after you 'dry' them a little.

Incidentally, your Indian pal who wrote "waa-laa" is corrupting the French word "Viola" which means "I used the play the violin but I got so excited at this discovery, I took up a larger instrument." It's a strange French colloquialism which doesn't translate into English too well.

SOFTWARE HOUSES: Good info. I hope I need it someday soon.

ASM: I wonder how many of use can AFFORD to save (?) $100 by using (not to mention WRITING) a different assembler. I say different, because I am not positive it is incompatible, but I bet it is. I don't mean to castigate David Whitman.I wrote one once. It WAS fun, for sure.

Final comments (yes, I'm tired of this, too): You need a spelling and grammar checker. Yes, my spelling is bizarre too, but it's intended in my case. Please forgive the negativeness above (where found); it's offered with supportive intent. The good work you are doing far outweighs the nits I have picked. Good luck. You might hear from me again. All you can do is hope.

Yours in bits and bytes,

Dion L. Johnson

SOFTWARE MAKES HARDWARE HAPPEN.

APPLE BUTTER:

It seems that IBM is working on a new console style computer called the "PEANUT." Apple on the other hand has finally introduced their new LISA computer. Apple is going up in price, while IBM is going down in price.

No doubt after IBM has announced their console computer, Apple will be having a bad case of peanut envy.

Remember, you heard it here first! /PJ

CARPET BRIGHT

"Hi, how are you today, need a rug shampoo?? We have a special offer this week for..."

The phone solicitation calls seem to increase when the nation's economy slows down. We consider our home a private place and such calls are very unwelcome. One day in particular after waking up from a deep sleep on Saturday morning to the sounds of "Carpet Care has a deal for You !!" I became very angry and spent 5 minutes telling the phone sales manager where he could put my phone. Later when I was still awake and burning at a dull simmer an interesting idea occured to me. DO YOU REALIZE THE POWER OF YOUR MODEMS? Imagine what 5 programmers with modems could do if they decided to express their dissatisfaction with a certain company? They could all set their modems for auto-dial and repeatedly dial the company in question for days on end. The company would never have a free phone line for quite a while. Answering 1000 carrier signals a day would be pretty depressing. I do not know if this is legal, but if old Carpet Bright can call me anytime, then turn about is fair play.

John Turner, Lawrence, KS

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOOLS

by Greg Estes

One of the primary foundations of a good program is a bullet proof input routine. A routine that controls the cursor and returns all the characters the user typed in.

In many discussions with other programmers about input routines the question of "what you like" determines a few aspects of this routine. Experienced programmers say "Well, I like it this way but I have found that most users like it another way." The difference is similar between a professional graphics artist and an amateur one. An amateur artist listens to your specifications for a while and then delivers a product that reflects their own taste with a few possible exceptions where your ideas were used. If you don't like the results you are told to go somewhere else because THEY like it the way it is. It would compromise their artist ethics to alter it. A professional gives YOU what you want and fits in their experience where it is appropriate. This is an over simplification but you probably know what I mean. An experienced programmer has had enough lumps put on his head by angry users to know that "good" is merely a reflection of the user. Computer software is definitely an architectural craft. Someone else is going to live with it for a long time. I am not giving you an input routine that I personally think is ideal but one that users have said they like.

The input routine is in BASIC. This makes it readily understandable and changeable, though it could just as well be in any language. To program it in assembly would cause undue hardship when an unusual change was needed for a particular application. Also the blinding speed of assembly is not really necessary in an input routine.

How do you like your screen input to be? What is a good overall input technique? Let's develop the SCOPE of our routine. What we want it to do and what we don't want it to do.

One philosophy about input routines is the "mini word processor" concept. The idea is to have ALL of the functions available (full cursor controls with insert, delete, scrunch , etc.) This is overkill in my opinion. Besides making the input routine quite large it complicates it substantially. Ask yourself, if you just typed in a 7 letter word incorrectly, is it faster to left arrow, insert, and right arrow back to your place OR to just delete the whole word and type it in again. Deleting the word and retyping is faster. So we are leaving the full "word processing" functions out of our input module.

The other enormous part of an input routine is the "acceptable characters" code. In a large input routine you can specify "A-Z, a-z" and that is all it will accept. But that sort of checking only does half of the job. We still have to validate the data when we get it back. So just gather the characters typed in by the user and do the error checking later.

Our method is to start at the first field on the screen (after arriving from a menu) and enter the fields from the top to the bottom. Once at the bottom we either want to ADD the record we entered because it was good, or change a field because it was wrong or try it all over again starting at the top. The other problem we have is finding ourselves at the 4th field and noticing that the 2 field is spelled wrong. Can we go up and down the screen at will? Can we blank out a field easily without having to hold down the space bar? We have to code in all of these features. Whether we do a good job has a tremendous influence on how our software is viewed as "user friendly."

Specifically our routine will indicate the maximum length of the field by dots. The user may use the BACKSPACE key to erase a previous letter. No insert is provided. The user may travel UP through the fields using the ESC key and travel DOWN through the fields using the RETURN key. This is non-destructive traveling. In other words as you pass through the fields, what was there previously will remain unless you enter something else. Entering an "*" in the first position of a field totally erases the field. Once at the bottom of the screen they are asked if they want to correct a field by entering the field number, or accept the whole works as valid, or erase the whole set of fields because they are incorrect. Exiting back to the main menu is provided by ESCing out of the first field in the screen. In general, hitting RETURN means everything is OK so keep going (ADDing), hitting ESC means backup to the previous field or eventually back out to the main menu.

Begin your program with the appropriate initialization. This can be done in a subroutine.(Be aware if you use the BASIC compiler that you must physically define a function before you use it.)

```
10 '***** MAILING LIST PROGRAM ********
20 GOSUB 500' INITIALIZATION
.
.
.
500'*** INITIALIZATION ******
510       TRUE=-1:FALSE=0
520 SCREEN 0,0,0:COLOR 7,0:FOR I = 1 TO 10 :KEY I,"":NEXT I
530 DEF FNCAP$(A$)=CHR$(ASC(A$)-32):CTLB$=CHR$(3)
540 BS$=CHR$(8):RTN$=CHR$(13):ESC$=CHR$(27):MES$=""
550 RETURN
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

We will be using all of the variables in line 540 for our routine.

The logic of our input routine is this: keep collecting characters that the user types in (WORD$) and when they hit the RETURN key it means they are done and want to move on to the next field. Once all of the complex cursor and input functions have been isolated into one module we can write just a few lines of code to get any variable. This is one of the primary reasons for putting so much effort into such an input routine. For every variable we need only code the following lines.

```
2500 MES$="Enter last name only"
2510 R=10 : C=15 : WD$=LASTNAME$ : WL=18 : GOSUB 1000
2520 IF BACK THEN GOTO .......previous field
2530 IF WD$="" THEN GOTO 2510  (error checking, try again)
2540 LASTNAME$=WD$
```

Notice how clear this makes our programs. Our routine will handle a prompt MESsage if we need one. It then determines the ROW (R) and COLUMN (C) to be passed to the routine. WD$ is the variable we use to pass the PREVIOUS value of this field, either it has been initialized as a blank field or if it is an old record then it will contain the old value. The WL indicates the maximum Word Length of the field. Then we go to the input routine starting at line 1000. (If you are still using the BASIC interpreter editor to write your programs then it is a good idea to begin all major pieces of code with a comment line because you tend to erase them when you are listing.) We will explain each line of code before we see it in its entirety.

```
1000 '******INPUT ROUTINE ****************
1010 LOCATE 25,1:PRINT MES$;
        Display the prompt message on the bottom line of the
        screen.

1020 LOCATE R,C,0:PRINT STRING$(WL,46);:WZ=0:WORD$="":BACK=FALSE
        Locate to the field position, display a line of dots
        equal to the Word Length, (then initialize the variables
        we need in the subroutine)  WZ= the number of characters
        we have collected so far, WORD$=the storage area for
        characters. BACK will tell us that the user wants
        to back up to the previous field.

1030 LOCATE R,C+WZ:IF WZ<WL THEN PRINT CHR$(95);
        After a character is trapped from the user it is
        located to the next position on the screen. We have
        to display these manually because the INKEY$ statement
        does not do it for us. Then an awkward case is handled
        where the user tried to backspace to much.
        (Character 95  is the cursor character "_".)

1040 INK$=INKEY$ :IF INK$="" GOTO 1040
        Here is the normal "get a character from the
        keyboard" loop. If no character is received from the INKEY$
        function then it just endlessly loops.

1045 ' IF INK$=CTLB$ THEN CLS:SYSTEM
        If you want to trap control break then use this
        line. It will take you back to DOS if a control break is
        pressed. (Or say ...THEN GOTO 1040  to totally trap.)

1050 IF ASC(INK$)=0 OR ASC(INK$)=9 THEN GOTO 1040
        Check for unwanted keypad characters, ( they return two
        values, the first of which is 0,) or the TAB key. If
        either then skip a bunch of code.

1060 IF INK$<>ESC$ THEN GOTO 1090
        If the key pressed is NOT the ESC key then skip the
        next 2 lines.


1070 BACK=TRUE
        We know they pressed the ESC key so set the BACKup
        variable to TRUE so when we return we know that the
        user wanted to backup

1080 LOCATE R,C:PRINT WD$;:PRINT SPC(WL-LEN(WD$));:GOTO 1200
        Print what we had in WD$ to start with, then
        print spaces to the right of the word to erase the dots. We
        do this by subtracting the len of WD$ from the
        maximum Word Length. Then goto 1200 to end it all.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
1090 IF WORD$="*" THEN WD$="":LOCATE R,C,0:PRINT SPC(WL);:GOTO 1200
       We jumped to here if INK$ was not an ESC. Now we
       check to see if the FIRST letter of WORD$ is a "*".
       if it is then we know that the user wants to erase
       the whole field. If it is, then we set WD$
       to nothing, then locate and print spaces to
       erase the whole works, then goto the last line.

1100 IF INK$<>RTN$ THEN GOTO 1130 ELSE LOCATE R,C
       If INK$ is not return then skip to 1130 otherwise
       setup for the next lines by putting the cusor back
       to the beginning position in preparation for
       displaying the final WD$ because the user is done.

1110 IF WZ=0 THEN PRINT WD$;:PRINT SPC(WL-LEN(WD$));:GOTO 1200
       If WZ=0 then we know no characters other than RETURN
       were entered and the user is just traveling down
       the screen . This means the user is keeping the
       original value of the field. so display it, and erase the
       dots and then goto the last line.

1120 WD$=WORD$:PRINT WD$;:PRINT SPC(WL-WZ);:GOTO 1200
       We know the use has hit RETURN and there is
       something in WORD$ (our collection variable)  so set
       the returned variable WD$ equal to what we collected,
        erase the dots, and goto the last line.

1130 IF INK$<>BS$ AND WZ=WL THEN BEEP: GOTO 1040
       If the user has not hit a Backspace (BS$) but the
       length of what we collected (WZ) is equal to the maximum
       Word Length (WL) allowed then BEEP and ask them for
       something else by going to 1040

1140 IF INK$<>BS$ THEN GOTO 1160 ELSE IF LEN(WORD$)<1 THEN GOTO 1030
       If the user has not hit backspace and we have
       thrown out everything else then it is something we want to
       save so goto 1160 and add it onto the end of WORD$.
       Otherwise, at this point, we know it is a backspace so
       see if they have backspaced all the way to a length
       of zero, if they have then goto 1030 and try again.

1150 WORD$=MID$(WORD$,1,LEN(WORD$)-1):WZ=WZ-1
       We know that INK$ is a backspace, so take our collection
       variable (WORD$) and subtract one character from it, then
       subtract 1 from WZ which tells us how long the  WORD$ is
       (and WZ tells us where to print the new character
        received on the screen.)


1155 IF WZ=WL-1 THEN LOCATE R,C+WZ:PRINT "_";:GOTO 1030
       Here we cover the awkward case where they have
       backspaced from the maximum length. Usually we move
       the cursor over one for a backspace and print a "_."
       but here we can only print a "_"  then go back to 1030
       and get the next character,

1156 LOCATE R,C+WZ:PRINT "_.";:GOTO 1030
       this is the usual backspace thing to do, move the cusor
       back one and print a "_." then go back for more.

1160 WORD$=WORD$+INK$:LOCATE R,C+WZ:PRINT INK$;:WZ=WZ+1 :GOTO 1030
       This is the normal entry case. They have inputed a good
       character and we add it to our collection pot (WORD$), we
       then locate exactly where the new character should be
       and then print it, add one to the WORD$ length (WZ)
       and go back for more.

1200 LOCATE 25,1:PRINT SPACE$(LEN(MES$));:MES$="":RETURN
       This is the final return line from the routine. We
       erase the MES$ if there was one, set it equal to
       nothing and then return to the main program.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

That is it for our input routine. We won 't ever have to write it again (thank goodness,) but we may alter it from time to time for unusual programs.

By far the best way to understand an input routine is to start with a small one like we mentioned last time and step by step add on the features you want. It seems a little too overbearing to try and explain these sorts of modules line by line. One of the more informative approaches to seeing what the INKEY$ statement gives you is to set up a loop like this and see what you get back.

```
10 INK$=INKEY$ : IF INK$="" THEN GOTO 10
20 PRINT INK$
30 PRINT ASC(INK$)
40 GOTO 10
```

Visually you can see what the function and numeric keys return to you. Remember that taking the ASC val of something only deals with the first character if INK$ is more than one in length. Also if nothing is there then it will bomb off (erase the second half of line 10 and see what happens.) /PJ

(Next time we will discuss the "3 deep" method of menu planning. There is a better way to handle moving around within a system. Would you believe directly jumping ANYWHERE in one step, or back to the main menu? No more intermediate backing in and out of 3 car garages.)

```
200 MES$="Enter LAST NAME"
210 R=7 : C=20 : WL=20 : WD$=LASTNAME$ : GOSUB 1000
220 IF BACK THEN RETURN
240 IF LASTNAME$="" THEN GOTO 200
250 LASTNAME$=WD$
260'
300 MES$="Enter ADDRESS"
310 R=9 : C=20 : WL=30 : WD$=ADDR$ : GOSUB 1000
320 IF BACK THEN GOTO 200
340 ADDR$=WD$
350'
  .
  .
  .
  .
  .
1000 '******INPUT ROUTINE ********
1010 LOCATE 25,1:PRINT MES$;
1020 LOCATE R,C,0:PRINT STRING$(WL,46);:WZ=0:WORD$="": BACK=FALSE
1030 LOCATE R,C+WZ:IF WZ<WL THEN PRINT CHR$(95);
1040 INK$=INKEY$ :IF INK$="" THEN GOTO 1040
1045'IF INK$=CTLB$ THEN CLS:SYSTEM
1050 IF ASC(INK$)=0 OR ASC(INK$)=9 THEN GOTO 1040
1060 IF INK$<>ESC$ THEN GOTO 1090
1070 BACK=TRUE
1080 LOCATE R,C:PRINT WD$;:PRINT SPC(WL-LEN(WD$));:GOTO 1200
1090 IF WORD$="*" THEN WD$="":LOCATE R,C,0:PRINT SPC(WL);:GOTO 1200
1100 IF INK$<>RTN$ THEN GOTO 1130 ELSE LOCATE R,C
1110 IF WZ=0 THEN PRINT WD$;:PRINT SPC(WL-LEN(WD$));:GOTO 1200
1120 WD$=WORD$:PRINT WD$;:PRINT SPC(WL-WZ);:GOTO 1200
1130 IF INK$<>BS$ AND WZ=WL THEN BEEP: GOTO 1040
1140 IF INK$<>BS$ THEN GOTO 1160 ELSE IF LEN(WORD$)<1 THEN GOTO 1030
1150 WORD$=MID$(WORD$,1,LEN(WORD$)-1):WZ=WZ-1
1155 IF WZ=WL-1 THEN LOCATE R,C+WZ:PRINT "_";:GOTO 1030
1156 LOCATE R,C+WZ:PRINT "_.";:GOTO 1030
1160 WORD$=WORD$+INK$:LOCATE R,C+WZ:PRINT INK$;:WZ=WZ+1 :GOTO 1030
1200 LOCATE 25,1:PRINT SPACE$(LEN(MES$));:MES$="":RETURN
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

COPY PROTECTION

COPY PROTECTION

by Timothy S. Anderson

The scene is set in a poorly lit basement. A small group of what appears to be deranged social outcasts hoover anxiously over a computer. The air fills with excitement as the first illegal duplication of CLONECALC takes place.

We, as software authors, shutter at the above mental picture while our publishers acknowledge it. On the other hand, how many readers purchased their PASCAL compiler, VISICALC, or favorite text editor? Of course that is ok because we, as computer professionals, are going to use these "tools" to further the computing industry.

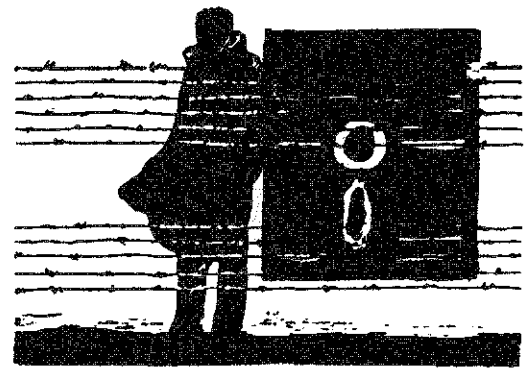
The unfortunate fact is that piracy has become a socially acceptable act, and will remain so until excellent software is sold at reasonable prices. Until then, the author can protect himself using some or all of the techniques presented in this article.

Probably the most effective means of protection is to tie your product to a piece of hardware that is sold with the software. Unfortunately, this financially complicates your venture. This approach is employed in products such as data communications systems, game controller adapters, etc., rather than data base managers or text editors.

All other types of protection attempt to make it not worth the while for the pirate. This might include the publishing of a large manual. Of course this can limit the appeal of your product by making it more cumbersome to use. Large manuals are effective with products used by technical individuals, ie. compilers, assemblers, debuggers, etc.

The most frequently employed methods encrypt the program in its storage medium. This is only a short term method of protection as eventually some one will discover your secrets. However, it is the quickest and most economical method of copy protection.

Encryption methods employed on the P.C. will always utilize the formatting of the diskette. Diskettes formatted and read by DOS rely on 40 tracks; each track is composed of 8 sectors; each sector being 512 bytes in length. Encryption takes place during formatting by writing track and sector identification fields on one or more tracks in a "secret format" . Perhaps track 9 is formatted as two 2048 byte sectors or maybe track 27 isn't formatted at all. The program being protected would then examine this track before it starts to run. In



practice, the actual loading of the protected program will require successful inputs from tracks or sectors that are uniquely formatted using the idiosyncrasies of the disk controller circuitry. Formatting tracks to your specification is straight forward, take a look in the Hardware Reference Manual, BIOS interrupt call 13. This method will hinder the casual pirate until he purchases a "disk backup utility" .

The backup utilities available today analyze the source diskette's formatting by reading the disk's track and sector I.D. fields. These programs contain their own drivers for the disk controller hardware, as DOS does not support this operation. Once the address of each track and sector are read, a duplicate image is written to a blank diskette. The only way a program such as this can be fooled is to know something about the disk controller hardware that the backup program doesn't.

The designer of the disk protection algorithm is in a constant battle with these utilities. The algorithm will undergo constant change as the latest backup utility becomes available. This cowboy felt his first protection algorithm was certain to end this madness. Before the algorithm could be put into service there was a backup utility on the market that could beat it. The battle had begun. The real protection is not of the program being sold, but of the protection algorithm itself. Every author can employ some of these simple tricks even if he doesn't elect to use a copy protection algorithm:

1. Use a loader to load a loader, of a loader, of a loader,... before actually bringing your program into memory. This will test the patience of the pirate while he attempts to disassemble your code.
2. In both the loaders and your program, intercept the DOS timer interrupt and periodically zero out the DEBUG breakpoint and single step vectors.
3. If your program doesn't use the serial communications facilities, don't forget to periodically change the communications link characteristics. A file can also be duplicated via a COMM port.
4. Make the operation of your program dependent on a hidden file - be sure to inspect the directory sectors to insure it is still hidden.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

5. Sprinkle your program with tests of the above mentioned conditions. Whenever one fails - idle the machine by disabling interrupts and setting in an endless loop.
6. Whenever possible, compile your programs. This is especially true with PASCAL. You can protect your idea, not your code, by the fact that there are few people who can understand the compiler's generated code.
7. Include plenty of warnings, copyright notices, etc., in the program. Be sure to mention the penalties of copyright infringement.
8. Leave misspelled words or nonexecutable code in the program, ie., two RET instructions following each other. There have been court cases that were won only because these mistakes appeared in both the original and the counterfeit.
9. Price your product so that there is a definite advantage to purchasing it rather than copying it. Provide a means of free updates by using a registration card.

As designers of software for the P.C., we are fortunate that the majority of these machines are being sold to corporations. These institutions will frown on illegal duplication using their equipment. Still, I have seen more copies of VISICALC than of honest purchases. How many copies do you think they could have been sold for fifty dollars? The most favorable copy protection is none at all. The only way to eliminate this problem is to restructure the entire business of selling software. There are too many middle men and not enough competitive products. □

If you are in the software marketing business and would like to use a copyprotection algorithm, one is available from the author. The algorithm is available on a royalty basis of 25 cents per diskette.

Timothy Anderson
Logical Systems Corporation
Route 1, Box 253
St. Michael, Minnesota 55376

Timothy Anderson has been in the computer field since 1975, working primarily with microprocessors as an electrical engineer. He entered the priesthood in 1978. Presently he is working as a consultant in microprocessor based product design, specializing in process control and data communications systems.

The Norton Utilities™ POWER TOOLS™
Your data is in *danger!*
if you don't have the file recovery routines *UnErase* and *FileFix* just two of the incredibly useful *Norton Utilities*

File recovery:
UnErase - recovers erased files
FileFix - repairs damaged files
Diskette display:
DiskLook - complete disk displays and maps
Diskette patching and modifying:
SedMod - easy changes to diskettes
Hidden file control:
FileHide - interactive hidden file control
BatHide - automatic hidden file control
Time control:
Timemark - displays date, time, elapsed time
Screen control:
ScrAtr - sets DOS to work in any colors
Reverse - work in black on white
Clear — clears the screen for clarity
Diskette file sorting:
FileSort - keeps files by date or name
DiskOpt - speeds diskette access
Sound:
Beep - sounds a tone to alert you
File printing:
Print - prints files

IBM buys The Norton Utilities again and again. Shouldn't you buy a set?
Available from ComputerLand, other dealers, and directly from me for $80:
Peter Norton 2210 Wilshire Blvd. #186 H Santa Monica, CA 90403

"Norton's many utilities have received almost **universal praise**, and have become **an industry standard**. No home should be without them." —**New York PC Club**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CHARACTER SETS

INTRO TO USER DEFINED CHARACTER SETS

by Mark Minasi

This article discusses alternate character sets for the IBM Personal Computer. It suggests uses for alternate character sets, and describes how to implement them.

The Personal Computer uses an enhanced ASCII character set. ASCII itself is only defined for code values 0 to 127, but IBM allows the Personal Computer to refer to 256 codes, from 0 to 255. If you have never seen all of the characters, run the following short program (for an 80 character screen):

```
5 SCREEN 0:WIDTH 80
10 FOR I = 0 TO 255
20 IF I > 6 AND I < 14 THEN 40
30 PRINT CHR$(I);I;
40 NEXT
```

For a 40 character screen, show half of the set at a time:

```
5 SCREEN 0:WIDTH 40
10 FOR I = 0 TO 127
20 IF I > 6 AND I < 14 THEN 40
30 PRINT CHR$(I);I;
40 NEXT
50 DEF SEG = 0:POKE 1050, PEEK(1052)
60 PRINT "PRESS A KEY TO SEE 128 - 255"
70 A$ = INKEY$: IF LEN(A$) = 0 THEN 70
80 FOR I = 128 TO 255
90 PRINT CHR$(I);I;
100 NEXT
```

For those who are wondering why we skipped characters 6 to 13 (with line 20 in both programs), these are special characters and would erase some characters from the screen. They cause the Personal Computer to tab, return the carriage, home the cursor, etc. To see more about the IBM-supplied character set, read appendix G in your BASIC reference manual.

The last 128 characters are only available with the Personal Computer in the text mode (screen 0). If you run the above program and change the "SCREEN 0" command to 'SCREEN 1" or "SCREEN 2" , you will see random characters and/or blanks for characters 128-255. (Before we go any further, let me stress that this only works with the color/graphics board. If you have both graphics and the mono board, switch your output to graphics before trying this. Unfortunately, none of this is available to those with only the monochrome board.)

The reason for this is that screens 1 and 2 (graphics) require a user-supplied character table defining characters 128-255. IBM has configured the PC so that the user can specify his or her own character set for character codes 128-255. (Actually, for those who don't mind messing with the operating system, all 256 characters can be redefined.)

BACKGROUND ON GRAPHICS:

The Personal Computer requires 64 bits (8 bytes) to define a single character. Since there are 128 user-definable characters, 8 X 128 = 1024 bytes must be set aside in user memory for a character table.

The Personal Computer knows where this table is by looking at the address contained in locations 124,125,126, and 127 (decimal; note that all values and addresses in this article are in decimal and are absolute unless otherwise specified). The address of this character table is expressed as a 'segment' and an 'offset':

Location (decimal)	Contents
124	Low byte of segment
125	High byte of segment
126	Low byte of offset
127	High byte of offset

To put this address into the Personal Computer, use the following code:

(Assume variable BOTTOM contains the address of the first byte of the character table.)

```
1000 OFFSET=INT(BOTTOM/65536)*4096
1010 SEGMENT=BOTTOM-OFFSET*16
1020 BASE(2)=INT(SEGMENT/256)
1030 BASE(1)=SEGMENT-256*BASE(2)
1040 BASE(4)=INT(OFFSET/256)
1050 BASE(3)=OFFSET-BASE(4)*256
1060 FOR I=1 TO 4:POKE I+123,BASE(I):NEXT
```

Each character is defined as an 8 by 8 matrix of bits (see figure 1 for example). Each bit represents a pixel. (Pixels are the very small dots on your screen that make up characters.) If the bit is 1, the pixel is "lit," or "on." If the bit is 0, the pixel is "off." A capital "A" can be formed by "turning on" some of the cells in the 8 by 8 matrix, and leaving others off. In figure 1, you can see the exact pattern used by the Personal Computer to form the "A."

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

What if you wanted to define this "A" yourself, from scratch? First, the character is sketched out in an 8 by 8 matrix, probably on graph paper. Then all "lit" blocks are considered 1's, and all "off" blocks are considered 0's. Each row is then interpreted as a byte, as it is 8 bits long. The byte must be converted to decimal so it can be POKEd into memory. In the "A" example, the eight bytes are

figure 1

binary	decimal
00110000 =	48
01111000 =	120
11001100 =	204
11001100 =	204
11111100 =	252
11001100 =	204
11001100 =	204
00000000 =	0

Second, we would determine where the bytes should be POKEd. Suppose we want to define this "A" as code 200. (Recall that we can only define codes 128-255.) Suppose also that our base is 45000. Each character requires eight bytes, so number 200 starts at

((200 - 128) X 8) + 45000 = 45576

Then, we would POKE the values 48,120,204,204,252, 204,204, and 0 into locations 45576 to 45583.

When you want to use the character, use the CHR$ function to display it. For example, suppose we have defined a "tank" character, for some game, in code 190. To display it in position (x,y), LOCATE Y,X:PRINT CHR$(190);. Some figures may require combinations of two or more characters. A large battleship may need one character code for the left half and another for the right half. When this happens, you can put one or several of your special characters into one string variable, rendering your programming task easier. For example, you might have a game with very large battleships, requiring three characters: one for the rear, one for the middle, and one for the front; suppose they are in codes 200, 201, and 202. Rather than having to program

```
PRINT CHR$(200)+CHR$(201)+CHR$(202);
```

you can instead set up a string variable A$ early in your program:

```
A$= CHR$(200)+CHR$(201)+CHR$(202)
```

and then just put

```
PRINT A$;
```

wherever you wish to locate the ship. This method will also execute faster. The IBM also supports the SCREEN function (pages 4-255 and 4-256 in your BASIC manual) for user defined characters. This means that you can check for collisions in graphic games simply by reading the SCREEN function at a particular location.

You may not want to define an entirely new character set; you may simply want to modify the IBM set. The task is simplified by the fact that the BIOS ROM contains the character table for codes 0-127 (alas, there is no table for 128-255) in locations F000:FA6E to F000:FE6E. (The 'SSSS:OOOO" format is often used when referring to addresses in the PC. It refers to a Segment—" SSSS" and an Offset—" OOOO" . To find the absolute location in memory, add 16 times the segment to the offset.) It is then simple to copy that into RAM, set locations 124-127 to refer to it, modify it, and use it for your own character set (again, for codes 128-255).

USES:

Graphics have many uses in simulation, games, foreign language programs, etc. Some ideas:

a) Oversize Symbols can be constructed of two or more characters. For example, an integral sign, or large parentheses.

b) Foreign Symbols can be defined. For example, a Russian language program could print Cyrillic symbols.

c) Reversed Symbols can be defined from the letters presently available on the Personal Computer. A set of reversed letters could produce "mirror writing."

d) Table Construction can be assisted with oversize letters, boxes, etc.

e) Alternate Character Fonts like serif/sans serif, black lettering, italic, script.

f) Gaming Characters for video games: spaceships, cars, animals, animation.

g) Superscripts, Subscripts for equations or footnotes.

h) Inverse Video Characters is supported in the text mode, but not in the graphics mode. You could create an inverse font for use in the graphics mode.

This is certainly not an exhaustive list, but only a start.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STORAGE OF THE CHARACTER SET:

When I wrote the Character Set Definition System, my biggest problem was finding 1K of RAM to put the character table in that wouldn't destroy anything or be destroyed by something else. I have 256K, so I was able to use high memory for development work; but this wouldn't be of much use to someone writing games on his or her PC with 48K —BASIC and DOS use all that space. In my System, I used a fairly complex method to find sure-fire, undamageable safe space; however, given the tutorial nature of this article, I have saved it for last.

One method to set aside space is the CLEAR function in BASIC. It tells BASIC exactly how much space to use for it's data, i.e. your program. I have found CLEAR to be a little unpredictable; it works, but test different values for the CLEAR until you find one that works. Then load your character set to the top of your memory.

User-defined characters give the PC the ability to do an amazing amount of things. They have the double virtue of making programming easier for the author and more effective from the perspective of the end user. Good luck and happy computing to all who use them.

EXPERT'S NOTE: how I controlled space in the Character Set Definition System.

The following is something of a lengthy explanation of how I set aside space within BASIC for machine code or a character table or such. It is not meant to be opaque, but one should have a nodding acquaintance with phrases such as "data segment," "assembler," and the like. This also involves a little assembler.

I used an old trick to set aside space: I dimensioned an array, then didn't explicitly refer to the array at any point in the program. This way, I have a safe section of memory to place things in. I needed 1024 bytes, so I dimensioned TBL-(150). This is double precision, and so sets aside 150*8=1200 bytes (I was sloppy). Now I have to find out where it is.

The VARPTR$ function in BASIC will give you the location of any variable as an offset to BASIC's data segment. To find this, use the line:

```
OFFSET = CVI(MID$(VARPTR$(TBL#(1)),2,2))
```

This is sufficient in some applications, like loading machine code; all BASIC needs is the offset. But in the case of absolute addresses (such as you need for the alternate character table), you also need to know the value of BASIC's data segment.

The BASIC manual notes that when a machine code subroutine is called, the DS register contains the value of BASIC's data segment. This then can be found with the following lines of assembler:

```
cseg segment
     assume cs:cseg
subrt proc far
     push bp      ; save bp. manual says to do this.,
mov bp,sp
     mov si,[bp]+6 ; get address of A%
     mov ax,ds     ; copy DS value into AX
     mov [si],ax   ; store into A%
     pop bp
     ret 2
subrt endp
cseg ends
    end
```

To use this, assemble and link it as the manual suggests, then call it from inside your program. A% will have the value of the data segment. To find the absolute,

```
ABSOLUTE = OFFSET + 16* A%
```

This value can be used as the bottom of your character table. □

PARDON ME, MODEM.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CHARACTER GENERATORS

Character generator programs serve a number of purposes. The primary function is to allow you to easily create your own symbols. Whether you use those symbols for letters in an alphabet, spaces ships or scientific notations is up to you. They are a much needed and often overlooked accessory to a programmers arsenal of tools. If you happen to be writing educational software then designing a number of figures to enhance your screen layouts would be quite handy. As a game programmer you can quickly design a set of alien army figures to attack the hero. Or if calligraphy excites you then sculpting a custom type font may be your cup of tea.

All of the normal ibm letters and special characters are set aside into 8x8 boxes. To redefine them you just need to make up your own 8x8 set of bits and push them into the right spot. Or better still make up a whole new 255 set and store them somewhere in memory. The concept is simple, but implementing it yourself is involved.

Mark Minasi's *Character Set Definition System* saves you 90% of the work. The system allows you to redefine the last 128 characters or a full 255. Once this has been done you can merge the appropriate BASIC subroutines provided to access them in your software. The Definition System allows you to quickly create the characters or letters you want one by one, then save them to a table in memory and finally to a file. You can reload and alter a previously defined set or just change the upper 128-255 ASC character set leaving the standard IBM ones intact.

The main program in the Definition System shows you an enlarged square with 64 boxes in it. You set the desired squares on or off to describe a character or a piece of a larger character. As each one is defined you save it to the table with a particular ASC number (128-255.) After you have completed as many as you wish then you save the whole works to disk giving it a distinct file name. Editing is as simple as reloading the character file and changing the symbols one by one. You can rotate any box, reverse video the bits, make a mirror image of the symbol and so on.

Once you have defined a set of characters, either 128 or a full 255, then you need to merge 10 lines of supplied code into the program that will use them.

The *Definition System* is definitely not for the inexperienced novice because you have to be generally familiar with programming and merging code. The documentation and source code (all the programs are in unprotected BASIC) are a good technical reference for illustrating the process of handling the character sets in the IBM PC. If you are planning a game that will need a number of predefined pieces, have a client that would like a new alphabet setup, need a large number of 'legend" symbols for charts, or you want an increased knowledge of how the IBM deals with its text symbols then the *Character Set Definition System* will speed you on your way. The final use of the sets is up to you and your program. /PJ

TOTAL

CHARACTER SET DEFINITION SYSTEM

Now for all 255 Characters !

IBM says you can only define characters for CHR$(128) to CHR$(254). This program lets you redefine them ALL from CHR$(0) to CHR$(254) — *create a COMPLETELY NEW SET*

Create your favorite type fonts: *Italics*, Old English, Block, or something totally meaningless for encryption purposes. Speed up your game graphics, design foreign character sets, reverse/inverse normal text, built double size letters, or special business and scientific symbols.

128 Character System (48K required) $50
Full 255 Character System (96K required) $75

Complete with documentation, demo programs, sample character fonts, and tutorial instructions.

(Data & Documentation sheets only . . . . . $25)
(Can be applied to later purchase of full system)

Send to : Mark Minasi
1204 N. Rolfe St. #1
Arlington, VA 22209
ph. (703) 276-0170 (evenings)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ACCESSING THE BIOS FUNCTIONS FROM C86

by James E. Randall

One of the attractions of the C language is the ease with which it interacts with machine hardware at a level approaching that of machine language. After acquiring the C86 compiler from Computer Innovations, Inc. (CI-C86), my first project was to write a set of graphics functions which plot the results of computations, Such functions serve purposes similar to PSET, LINE, and other BASIC commands.

This article presents these graphics functions as a tutorial example of interacting with the basic input/output functions (BIOS) in the Personal Computer ROM. As will be pointed out, the generality of the BIOS functions makes them slow for performing repetitive operations, such as plotting lines.

Appendix A in the Technical Reference manual contains a listing of the BIOS software. The accompanying documentation describes the routines which are executed whenever specific interrupt instructions are given to the microprocessor. As an example, the instruction INT x10 transfers control to the routine pointed to by the contents of addresses x40-43. The routine, called VIDEO—IO, starts at absolute address xFF065 (in ROM) and is listed starting on page A-43 of the manual. This software is one way of interacting with the video displays, with the desired operation being determined by the contents of the 8088 registers at the time that the INT 10 instruction is executed.

The CI-C86 Library contains the function sysint() which executes an interrupt instruction after setting the 8088 registers to specified values. For my application, one C function was defined to initialize 640 by 200 black and white graphics by executing INT 10 with register AH=0 and AL=6, as described in line 3131 on page A-43. Another C function sets the registers as required to place a dot at some given X and Y coordinate on the CRT screen. The ROM software takes care of locating the particular bit and byte in memory which correspond to the indicated screen coordinates.

The library function sysint() has three arguments: the interrupt to be executed (x10 for video), a pointer to a structure containing the initial settings for the 8088 registers, and a pointer to another structure containing the values of the 8088 registers upon returning from the video processing routine. (The latter locations are used when the video routine tests whether a dot is present at given coordinates.) The sysint() call contains advanced concepts which required more explanation for this novice C programmer than that given in the CI-C86 documentation.

Arguments in FORTRAN subroutines and functions are passed as the address containing the argument, a pointer. Thus the argument itself is common to both the calling routine and to the called routine. In C the arguments are actual values which are placed upon a stack by the calling routine and then deleted upon the return. Thus they are of a temporary nature. The called function can not directly modify the value of a variable in the calling function.

C functions can have pointers to variables as arguments by using the ampersand operator & as a prefix to the name of the variable. The function sysint() supplied by Computer Innovations, can be used to call the VIDEO—IO routine in BIOS (INT 10) by the statement

```
sysint(0x10,&srv,rrv);
```

in which 0x10 is the interrupt to be executed, &srv is a pointer to the structure srv where the desired sending register values for the 8088 have been placed, and &rrv is a pointer to the structure rrv where the returned register values are to be found. The structures of srv and rrv and their contents must have been previously set according to the purposes of the call to the video routine.

The library function sysint() expects srv and rrv to have a specific structure consisting of 8 integers, each corresponding to one of the 8088 registers. Thus before sysint is called it is necessary to 1) declare a general structure format having 8 integers, corresponding to the 8088 registers: ax, bx, cx, dx, si, di, ds, es; 2) define in memory two such structures having this format, srv for the sent register values and rrv for the returned register values; and, 3) set any of the 8 register which are to be

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

used by the interrupt called. The video routine (INT 10) only uses registers ax, bx, cx, dx; so I defined a function int10(ax,bx,cx,dx). This function passed the values to go into the four 8088 register, moved them into the structure srv and then called the function sysint().

The accompanying listing is for a program to draw a box on the high resolution graphics screen. It illustrates how to set the 8088 registers for a call to one of the BIOS functions. The functions 1) execute interrupt x10 with registers ax-dx set; 2) initialize for black and white graphics; 3) place a dot at coordinates X,Y; and 4) draw a rectangle at given coordinates.

The opening line establishes the format of a structure, here named as regval consisting of 8 integers, named to correspond to the 8088 registers involved. The second program line establishes two such structures in memory, srv which will contain the sent register values, and rrv which will contain the returned register values. These structures are global, accessible by functions in the program.

The main program uses two functions, one to initialize the black and white graphics mode, the other to draw a rectangle using the given two diagonal corners at 50,100 and at 150,150. (The rectangle appears as a square on my display monitor.) The balance of the program consists of a collection of functions, characteristic of the way C programs are structured.

The function int10() places four integer arguments as required by the 8088 registers for the interrupt routine. The function bwgrph() calls the int10() function with the high and low bytes of the ax register set as needed to initialize the black and white graphics mode. The dot(x,y) function calls the interrupt 10 function with register cx set to the X coordinate and register dx, set to the Y coordinate.

The rect() function draws first the horizontal lines, then the vertical sides of a rectangle. As written the horizontal lines are plotted only at every other X-coordinate. By doing this the horizontal and vertical lines appear to be of similar intensity and the function executes somewhat faster. Also, horizontal lines drawn with alternating dots and blanks do not exhibit a problem I have found in using the BASIC command to plot boxes, and in FORTRAN and C calls to the BIOS plotting function.

The problem with plotting solid horizontal lines is that the termination of the line as plotted on the screen is several pixels beyond the X-coordinate as located by PSET or vertical line commands. On the other hand, plotting horizontal lines with alternating dots and blanks appear essentially solid, but stop where they are supposed to. At this writing I have not resolved this puzzle.

Although I knew that there was considerable overhead associated with the BIOS calls I was surprised with the result of a time test for drawing 100 rectangles with the rect() function as defined here and compared with a similar operation performed with interpretive BASIC. The BASIC routine, which used the LINE (X0,Y0)-(X1,Y1),,B box command, ran twice as fast. Calls to BIOS routines are practical for single operations, such as setting the display mode, or establishing the baud rate of the communication port, but they are not effective where speedy plotting or communications are the objective. □

JUST LOOK AT YOU, HARRY. SOMETHING INSIDE ME JUST KNEW YOU NEEDED A VACATION; KNEW THIS WAS JUST THE PLACE TO GET YOUR MIND OFF COMPUTERS.
1150 N=5
1160 RETURN
1170 IF R>180 THEN 1170
1180 REM PROCESS LEFT NEEDLE
1190 IF R>90 THEN 1190
1200 IF R>10 THEN R=10
BINE

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
/*   Illustrates using int 10 for graphics functions.          */
/*       C86 function sysint() requires structure to hold      */
/*         8 integer values of 8088 registers at time          */
/*         interrupt instruction occurs.                       */
/*       Establish format of structure, called regval;         */
/*         define two such,  srv and rrv in memory.            */

struct regval { int ax,bx,cx,dx,si,di,ds,es; };
struct regval srv,rrv;

main()
{
        bwgrph();                   /* initialize B&W graphics   */
        rect(50,100,150,150);       /* 50,100 and 150,150 are    */
}                                   /*  diagonal corners of rect. */

/* Call to bios int10, register values as arguments             */
int10(ax,bx,cx,dx)
int ax,bx,cx,dx;
{
        srv.ax=ax;                  /* arguments moved into      */
        srv.bx=bx;                  /*    structure srv          */
        srv.cx=cx;
        srv.dx=dx;
        sysint(0x10,&srv,&rrv);     /* do the int 10 instruct.   */
        return;
}
/* upon int 10: ah=0, al=6 sets B*W high resol graphics         */
bwgrph()
{
        int10(0x0006,0,0,0);              /* hi=0 mode; lo=6,bw */
        return;
}
/* put dot at x,y; */
dot(x,y)
int x,y;
{
        int10(0x0c01,0,x,y);              /* ah=12; al=color    */
        return;                           /* cx=x; dx=y coordin. */
}
/* draw rectangle x0,y0 and x1,y1 as diagonal corners           */
/* dots at alternate x's for uniform intensity                  */
rect(x0,y0,x1,y1)
int x0,y0,x1,y1;
{
        int x,y;
        for (x = x0; x < x1; ++x)
              {dot(x,y0);           /* top                       */
               dot(x,y1);           /* bottom                    */
               ++x;}                /* skip one x dot            */
        for (y = y0; y <= y1; ++y)
              {dot(x0,y);           /* left side                 */
               dot(x1,y);}          /* right side                */
        return;
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

by Frankie Detesto

(There was a typo last time. To set the default drive to something other than the one you are currently on do a:

SET DEFAULT TO B:

You should become very familiar with the SET options on pages 128-133 in the B section of the manual.)

You can design very tasteful input screens using dBASE II. You have access to all of the special symbols that the IBM PC has stored in its character set. To demonstrate, create a program with the following code and then DO it.

```
ERASE
SET TALK OFF
STORE 15 TO X
STORE 1 TO ROW
STORE 1 TO COL

DO WHILE X < 256


STORE 1+COL TO COL
  IF COL > 39
     STORE 1+ROW TO ROW
     STORE 1 TO COL
ENDIF

STORE X+1 TO X
@ ROW,COL SAY CHR(X)

ENDDO
```

The SAY CHR(X) displays the symbols. It is important to note that whenever you want to draw a large box with a repeated symbol, it is faster to make up a big string containing one complete line of symbols and THEN put it on the screen. The I/O work need only be done once. Most of your screens will be box shaped anyway so you could create a standard BOX program and then DO it just before displaying the rest of your menu text. We will use the nifty SAVE/RELEASE feature. dBASE II is a little slow so we have to go for SPEED whenever possible. We can create a STRING variable made up of anything (in this case special box characters) and SAVE the variables to a memory file. Then whenever we need them we can use RESTORE without having to rebuild them. Run this next program (which we will call BOXBUILD.PRG) to create 2 strings containing the TOP and BOTtom of our box.

```
ERASE
SET TALK OFF
RELEASE ALL
STORE CHR(201) TO TOP
STORE CHR(200) TO BOT
STORE 0 TO X

DO WHILE X<69

STORE TOP+CHR(205) TO TOP
STORE BOT+CHR(205) TO BOT
STORE X+1 TO X

ENDDO

STORE TOP+CHR(187) TO TOP
STORE BOT+CHR(188) TO BOT

@ 1,5 SAY TOP
@ 5,5 SAY BOT
* this will show you what you have

RELEASE X
SAVE TO BOX
* save to a BOX file
```

We have built up the top and bottom strings that make up a nice looking box. Notice that it takes a while to do this. Then we saved them to a BOX.MEM file. When we need them we can just :

RESTORE FROM BOX

Our next program called BOX.PRG will display a tasteful box on the screen in a reasonably fast manner using the BOX file.

```
ERASE
SET TALK OFF
RESTORE FROM BOX
STORE 2 TO ROW

@ 2,5 SAY TOP
* this is the top of the box

DO WHILE ROW<20
         STORE 1+ROW TO ROW
         @ ROW,5 SAY CHR(186)
         @ ROW,75 SAY CHR(186)
         * this displays the sides
ENDDO
@ 20,5 SAY BOT
* this is the bottom of the box
RELEASE ALL
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now your dBASE II programs (or .FMT files) can quickly display nicely arranged screens anytime they want to by merely putting in the following command:

DO BOX

There are numerous other creative ways to use the SAVE function to build and reuse particular strings.

INPUT TYPES: consistency in your input techniques is quite important. And by this I mean whether you use the WAIT command or the ACCEPT command and whether you allow a single key stroke to answer a menu question or force the user to hit a return before any action is taken. I am a member of the HIT squad, meaning I prefer hitting return after any response or question. Too often I have ended up causing a program to run inadvertently. I would suggest having the

SET CONFIRM ON

statement in all of your programs including menu programs. This enables a user to SEE what they are about to do before the computer takes off and does it. What you see is what you get. Be sure and let them SEE it first. The same goes for field input statements. Make then hit RETURN before moving on to the next field. I do have other personal preferences like setting the BELL off and stopping the inverse video. Those are a matter of individual preference.

In general I always use the ACCEPT statement for getting information from the user. Never a WAIT statement which has that unfriendly "WAITING" message. You could still use it but do this:

```
SET CONSOLE OFF
WAIT TO OPTION
SET CONSOLE ON
```

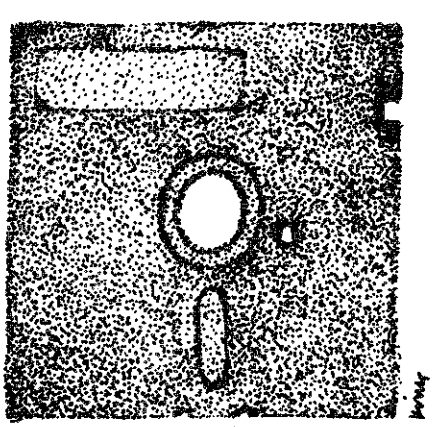
If you insist on using the WAIT command this is a good balance. It disables the WAITING message. You could use Pete Norton's SECMOD program and look at the piece of code in dBASE II and physically change the WAITING text to something else. Maybe overwrite it with "CHOICE" instead.

NESTING: You will notice in the dBASE II manual that the middle section is a "working accounting system." I have not tried it. But it seems readily apparent that the system was written before dBASE II added their CASE statement. It is the over abundance of those ENDIF, ENDIF, ENDIF, ENDIF nests that are a dead give away. The CASE statement makes most of the IF nesting obsolete.

```
IF OPTION=1
   DO NEWMAIL
ELSE
   IF OPTION=2
      DO OLDMAIL
   ELSE
      IF OPTION=3
         DO CALCULATE
      ELSE
         IF OPTION=4
            DO CUTENEST
         ELSE
            IF OPTION>4
              ? "Invalid entry"
           ENDIF
         ENDIF
      ENDIF
   ENDIF
ENDIF
```

The above can be simply restated as:

```
DO CASE

CASE OPTION=1
     DO NEWMAIL

CASE OPTION=2
     DO OLDMAIL

CASE OPTION=3
     DO CALCULATE

CASE OPTION=4
     DO CUTENEST

OTHERWISE
     DO TOOBIG

ENDCASE
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SPEED: If you do not have a combo board with at least 256K of memory you are missing out on one of the primary deluxe features of the IBM PC. By using your RAMDISK or FLASHDISK software you can create a fake diskdrive in memory. Then once you have copied all of the dBASE programs onto it you can enjoy Formula One speed as it were. I set aside 128K for main memory and then make the rest of it into a RAMDISK (drive C:). I then copy all of my dBASE II programs onto it. If I am only going to look at, print out, or play with my data files, then I copy them onto the RAMDISK also. As long as I am not going to update or add any records then there is no reason to leave the .DBF files on the floppy drive.

If my session with dBASE will involve adding some records to a file then I leave the .DBF files on drive B: and run dBASE programs on the RAMDRIVE. The best of both worlds are yours with this setup.

I have found my RAMDISK to be so reliable that I am willing to risk a few things. Normally I will only add about a half an hours worth of records at a time. So I leave an empty NEW.DBF file on my dBASE program diskette. When I copy all the dBASE programs onto drive C: I also copy the NEW.DBF file over too. Then I add my new records, and check to see they are OK. After that I:

```
USE B:OLDFILE
GO BOTTOM
DISP
APPEND FROM NEW
GO BOTTOM
SKIP -10
DISP NEXT 10
USE
QUIT
```

I open the old master file on drive B:, then I go to the bottom and DISPLAY the record number to see what the current count is. After that I append from the NEW file, followed by a rechecking of the end of the OLDFILE to verify that the append worked. The last USE closes the file and I am all done.

UPDATE :The current version of dBASE II is 2.3D. If your version number does not match this then contact Ashton-Tate for an update.

I admire the forthrightness with which Ashton-Tate acknowledges their program bugs. When I received the new 2.3D release a "Technical Support" sheet was included listing the known bugs that were either corrected or going to be corrected. Often large companies choose to ignore or refuse to admit that there are any bugs in their packages. We all know they exist so why can't they own up to them? Ashton-Tate exemplifies what "good support" is all about.

BOOKS:

The two main tutorial books for dBASEII are:

dBASE II USER'S GUIDE by Adam Green
From SoftwareBanc, 661 Massachusetts Ave., Arlington, MA 02174, price $29, call (800) 451-2502

This is a hands-own learning booklet that takes you through all of the commands step by step. A fine addition to the dBASE II manual which teaches you by leaps and bounds.

The other book is brand new from Ashton-Tate:

EVERYMAN'S DATABASE PRIMER featuring dBASE II by Robert Byers
From Ashton-Tate, 9929 West Jefferson Blvd., Culver City, CA 90230, (213) 204-5570

This is a more detailed discussion of how a database works, how to set it up, lay out your files, and plan your system. It is written for the novice and is well done.

Ashton-Tate also supplies a listing of applications written by other dBASE users and programmers. It is called *Application Junction*. Basically a clearing house listing for the pre-written applications they know about. Everything from Financial to Engineering to Medical packages are available. Write them for a copy.

If there are particular areas of dBASE II that you would like to see discussed just let me know. Next time we will cover some more advanced techniques. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

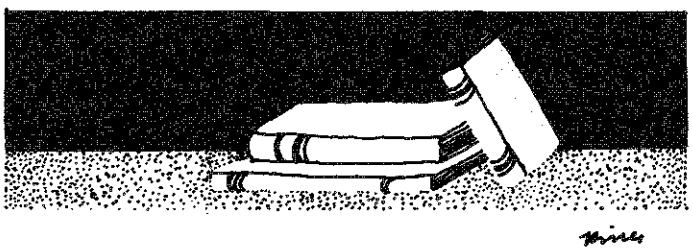
PJ LIBRARY

CHINESE PUBLISHING HOUSE:

Whether you write software or short stories you eventually have to face the inevitable rejection slip. Our computer expertise may exceed China's, but as far as refusal notices go their understanding of the human interface vastly surpasses our own. Below is a sample of how one says no in China. Assume you have just submitted a software article and this is your reply.

"We have read your manuscript with boundless delight. If we were to publish your paper it would be impossible for us to publish any work of a lower standard. And as it is unthinkable that, in the next thousand years, we shall see its equal, we are, to our regret, compelled to return your divine composition, and beg you a thousand times to overlook our short sightedness and timidity."

CREATIVE COMPUTING:
The March issue of Creative Computing has an excellent discussion of interrupts and flags by Will Fastie. Will writes the regular IBM PC column and it is well worth buying the magazine just for his articles concerning programming.



THE COMPUTERIST DIRECTORY:
Have you ever wondered what a telephone book for computers would look like? Well there already is one called the Computerist Directory. It includes a white pages and yellow pages for individual modem users (not specifically for the IBM PC though.) National on-line users associations, bulletin boards, clubs, etc are also listed. Write for one of their sample copies which are published twice a year.

The Computerist Directory
ALTERNET INC.
15250 River Road
Guernville, CA 95436
ph. 707-887-1857

THE INDEPENDENT:
There is a national newsletter for independent computer consultants. Send them a postcard and ask for a sample issue. Their subject matter surrounds the business and ethical standards a professional computer consultant should uphold.

The Independent
P.O. Box 27412
St. Louis, Missouri 63141
ph. 314-567-9708

LEGAL CARE FOR YOUR SOFTWARE:
If you ever plan to write and sell software, publish someone else's software, or are involved in legal software contracts of any kind you should purchase this book. The text is written by a lawyer that UNDERSTANDS what source and object code are, which seems to be a rarity. If you have sought legal software advice from you local attorney you know how frustrating it can be. Few general practice lawyers are familiar with software. 80% percent of your time is consumed explaining what source code versus object code is and YOU are paying for it. If you do not feel the urge to get this book for yourself, you might consider acquiring it for your lawyer. The topics covered are trade secrets, copyrighting, contracts, work-for-hire, evaluation and beta test agreements, license agreements, trademarks, patents, limiting liability, and many sample contracts. The book is called:

LEGAL CARE FOR YOUR SOFTWARE
General Books Division
Addison-Wesley Publishing Division Company, Inc.
Reading, MA 01867
ph. 617-944-3700
$19.95

THE C PUZZLE BOOK:
Alan Feuer, the author of the book, taught classes in C at Bell Laboratories. This book is a result of developing introduction course materials. Essentially it is a collection of pieces of C code. You are asked what each piece does. The answers explain the code so you can check yourself. This is an excellent book for improving your C knowledge on your own. definitely a companion book to *The C Programming Language* by Kernighan and Richie.

The C Puzzle Book
by Alan Feuer
Prentice-Hall, Inc.
Englewood Cliffs, NJ 07632

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SOFTWARE HOUSES

SOFTWARE HOUSE SUBMISSIONS

When it comes time to submit your software, looking professional is the name of the game. If you have ever worked for any publishing houses, software or otherwise, then you know the full spectrum of the submissions you receive. Try putting an employment ad in the newspaper and see the responses you get. Some are well done, others lead you to believe the person has talent but just can't express themself. Still others jot down a few remarks on the back of a McDonalds wrapper. I remember feeling pretty self-satisfied with my own resume until I happened by a type-setting shop and saw a PROFESSIONAL resume set in Garamond Light with Caslon Bold paragraph headings. A real enlightening experience. Next time you apply for a new job have your resume type-set. It adds a first class image. Thumb through your latest issue of BYTE or PC and compare some of the software ads. There are always a few that must have been designed by 3rd graders during a Halloween party.

From your first contact with a software house you began establishing an image. If you send in your software on a diskette with a few notes scribbled on part of a sheet of computer paper then you might as well forget the whole enterprise. Maybe if you have just written the next VisiCalc you will succeed in spite of yourself. But don't count on it. There are a few common sense ways to go about presenting a good image. These have been gleaned from Ted Lewis' manual on submissions, from conversations with software houses, and marketing people.

Every source program you submit should contain the following lines at the beginning of the code:

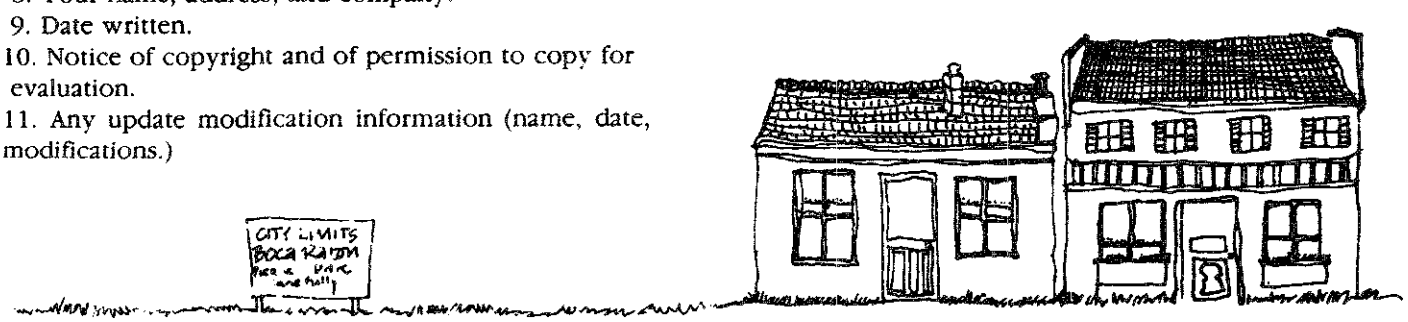
1. Program Name
2. A one line description of the program and its function.
3. The computer it is to be run on.
4. The configuration required (K, number drives, etc.)
5. The language used and version number.
6. Special hardware needed. (Paddles, printer, etc)
7. The files necessary to run the system.
8. Your name, address, and company.
9. Date written.
10. Notice of copyright and of permission to copy for evaluation.
11. Any update modification information (name, date, modifications.)

On every diskette include your name and address. Any documentation you send should also have your name and address.

One good way to keep in touch with acceptable practices is to read the book "Writer's Market" for book authors. If you follow all the rules for professional manuscript submissions you will not be too far wrong.

If you want to ensure that the software marketing firm will come looking for you with shotguns, just send them your super deluxe mailing list system with NO DATA FILES. That has got to be the Cardinal Sin of new software authors. How would you like to evaluate a mailing list system and spend the first hour typing in names and addresses? The next worst sin is sending a diskette with no startup instructions. Even some big name software packages do not give you hints about how to get to the main menu. No doubt it is there on page 78 but it will take you hours to find it.

Probably the hardest part of the software submission process is the selling of yourself and your program. As programmers we are used to the solitary, secluded life. Our "phone voice" and verbal PR abilities may leave much to be desired. If you have a talented friend who is expert at this sort of thing they may be a tremendous asset. A programmer or book author is too close to his own work. Just listen to a programmer describing his latest masterpiece, "It is a really good program, there are a few bugs of course that I haven't worked out yet. Trapping the out-of-bound errors stumped me for a while, but not too many users will type in 13 digits. Just ignore the misspelling of the ..." When you talk with a SW firm YOU are the salesman. If you recognize your own limitations in that regard then let someone do it for you. The match-maker philosophy is a good one. Go there in person if at all possible, but use the telephone if you can't. Once they like you they are closer to liking your software. /PJ



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

GRAPHICS IN BASIC

If you are planning to write a game for the IBM PC you undoubtedly printed out the DONKEY program and analyzed what they were doing. First you DRAW something in the upper left hand corner (or wherever) and then GET it into the appropriate size array. Once you have stored the sketch away you can PUT it anywhere you like to the screen. A very straight forward and simple technique. We all need to thank Microsoft for making this so easy. (They did saddle us with the 90 degree rotation problem, but then nothing is perfect.)

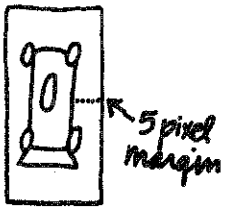
The new version of the BASIC manual discusses graphics in much more detail (figuring array sizes needed, etc.) If you have the older version of the BASIC manual (pre-July) you might consider purchasing a new one if you are serious about graphics. Here is what is stated about calculating the array size. X is the horizontal length and Y is the vertical length of the area to GET.

4+INT((X*2+7)/8)*Y

If you are using HIGH resolution (80 column BW mode) then the 2 should be changed to a 1. This equation gives you the number of bytes to allocate in your array. Notice when you DIM an array you can use INT, SINGLE, or DOUBLE precision. For example assume you are making a space ship and the size of X is 20 and Y is 10. Then the equation indicates that 54 bytes are required. With an array we set aside 2 bytes per element if it is INTeger and 4 bytes per element if it is SINGLE precision and 8 if it is DOUBLE. Any of the following would be suitable.

```
DIM SHIP%(27)
DIM SHIP!(16)
DIM SHIP#(7)
```

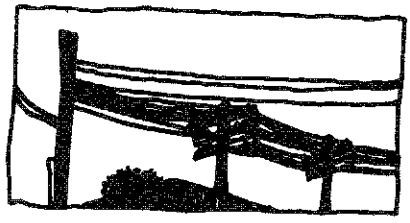
This part of the graphics setup is not too difficult. Decide the kind of animation you will need so you can reduce the FLICKER. This is a primary consideration. If you do not have to preserve the background over which the figure is moving then you are home free. Here is what to do. After you have drawn your space ship on the screen, decide how many pixels you will move it at a time, let us say 5 is the maximum jump at one time. Now when you DIM the array and GET the box containing the space ship allow a 5 or 6 pixel margin around the ship. Like this:



Then when you move the ship around the screen, you always PUT it with a PSET option. The margin you have allowed erases the old ship automatically. As long as you make your jumps equal to or less than the margin then you need not bother about erasing the old image. An interesting side effect is created when two figures run into each other that have been animated this way. /PJ

IBM PC MAINTENANCE CONTRACTS:

The well known computer service company SORBUS has entered into the IBM PC maintenance contract business. The large IBM mainframe owners that currently use SORBUS as their service contract holders can now just include their IBM PC's on the same agreement.



SHORT HAUL MODEMS:

Your local telephone company very probably provides a number of modem services. We recently checked into what they called a "Limited System" data set. We found a need to use a terminal at home connected to our mini-mainframe at work. All we needed was a terminal. But the thought of going 300 or even 1200 baud was depressing. We found out upon asking the local telephone folks (Bell, AT&T or who ever they are now,) that we could get a 19,200 synchronous data set for both ends and the cost was $36 a month. Not too bad. We did have to be within 3 miles of a switching station, but in most cities that is not a problem. There were other numerous options, so give Ma Bell a call if you need some more speed.

COBOL UPDATE:

This note from the IBM Users group in Portland Oregon. If you are using IBM's COBOL compiler you need to get an update. Take the pink first page of your manual to your dealer and they will send send in for the update at no charge to you. It may take several weeks to process.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOW TO PRINT-SCREEN FROM A BASIC PROGRAM:
(C) Copyright 1982 by Peter Norton.

When BASIC is operating on the IBM PC the "print-screen" function doesn't work. This article shows how your BASIC program can perform a print screen operation, and gives us an opportunity to learn more about how the IBM PC works.
Before we explain it all, here is how to print-screen in BASIC. First, you must set-up a print-screen routine, by executing these instructions, once only:

```
10 DEF SEG = 4095
20 POKE 0,205 : POKE 1,5 : POKE 2, 203
30 PRINT.SCREEN = 0
```

Then our programs can print the screen contents at any time by executing:

```
100 CALL PRINT.SCREEN
```

As long as the "DEF SEG" hasn't been changed, then the CALL statement will print the current screen display. This bit of BASIC program doesn't make the "PrtSc" key work — it lets your program perform the print-screen function. Your program must decide when to print the screen, perhaps by acting on a special function key.

To understand how this works we have to touch on some of the most fundamental parts of the IBM PC. Built into the read-only memory (ROM) are many service routines that perform frequently used operations, among them is the print-screen function. These services are activated by a mechanism known as a "software interrupt" , and when you request interrupt 5 you cause a print-screen to happen.

The programs which service the keyboard are also found in the ROM. The ordinary keyboard service program always checks for a "PrtSc" key : when this key is pressed, the keyboard program performs an interrupt #5, and the screen is printed. The keyboard service routine uses interrupt #5, but doesn't own it exclusively — any program which knows how to request interrupts can use it.

When BASIC is running it takes charge of the keyboard and so the regular ROM keyboard program doesn't see the PrtSc key, and BASIC ignores this key. This prevents us from using the PrtSc key to print the screen.

So, the print-screen operation is not available in BASIC ordinarily. But BASIC has the ability to accept and perform machine language subroutines. If we set up a program which will request interrupt #5 then we can CALL it and print the screen. That is what the BASIC program we have written accomplishes.

Our short machine language program consists of only two instructions: one to request interrupt 5, and the other to return to the caller. The interrupt instruction is 2 bytes long, and the return is one, for a total of three bytes. The three POKE statements above create those three bytes. The numeric values 205 and 5 are the interrupt #5 instruction, and the value 203 is the return instruction.

There is one little problem that occurs whenever machine language programs are used in BASIC — where to put them. There is no clean simple way to deal with this problem; it has to be solved differently depending upon the size of the computer's memory, and upon what else a BASIC program might do with "DEF SEG" related statements.

The example we have written is set up for a computer with 64K of memory and places the print-screen program in the last 16 bytes of memory. For another memory size, "X" expressed in kilobytes (or K), this formula will do the same:

```
DEF SEG = X*64-1
```

If X is 64K then this formula works out to 4095, which is the number used in the example at the beginning. If your programs set the "DEF SEG" for any other reason then it will have to be coordinated with its use for the PRINT.SCREEN program or any other such machine language program. When PRINT.SCREEN is called, DEF SEG must be set to the same value used when the program was created with the POKE statement. □

TELECOMMUNICATIONS TYPESETTING

COMPUTER OWNERS: If you have a modem, text editor, and communications program, you should consider the improved readability and cost savings provided by *TYPESETTING* your program documentation, lengthy article or manuscript, newsletter, or other text instead of just reproducing it from line printer or daisy-wheel output. Computer typesetting by telephone offers you high quality, space-saving copy that creates the professional image you want! Hundreds of type styles to choose from with 8 styles and 12 sizes "on line!" And it's easy to encode your copy with our designated typesetting commands.

CONFIDENTIALITY GUARANTEED
— bonded for your protection —
PUBLICATION DESIGN, EDITING, & PRODUCTION

EDITING & DESIGN SERVICES, INC.
30 East 13th Avenue Eugene, Oregon 97401
Phone 503/683-2657

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FREEWARE

FREEWARE

Free Assembler
David Whitman is offering a "free" assembler if you are interested in experimenting with assembly language programming. The assembler (CHASM) is written in BASIC. Send a formatted diskette with return postage to:

David Whitman
2 North Park St. Apt L
Hanover, NH 03775

Free Utility Programs
Remember the last time you went through your diskettes and decided to "clean them up" instead of forking over the money for another box of 10? What a royal time consuming task that was. Well weep no more. Jeffrey P. Garbers has saved the day by creating several utilities we all need desperately. And they are free too boot. One of them does a "delete by request." You merely type in a "DELETE *.BAS" and respond with a Y or N whether to erase a file. Instead of taking 5 minutes to clean-up a diskette it takes about 10 seconds. I usually do a "DEL *.*" and go for the gusto. His program called VDEL handles the deleting. Another one (FK) allows you to define the function keys differently for use under DOS. Now you can hit F1 and have it do a CHKDSK or whatever. Send a formatted diskette with return postage to:

Jeffery P. Garbers
890E Clubhouse Circle West
Decatur, GA 30032

Deal of the Year
If you want to get the Utilities mentioned above and several others (63 programs and documentation files to be exact) just send $5 (no diskette required) to:

Art Bevilacqua
14 Arthur St.
Danvers, MA 01923
(Make SURE you mention you would like UTILITY DISK #6, and you might request a list of their other diskettes. Art is the Disk Librarian for the Boston Area IBM PC group.)

Free Communications Software
PC-TALK is a wonderful piece of software for using the Hayes Smartmodem. Programmer's Journal sends all of our articles to the typesetter using it. For a copy of this "free" program send a formatted diskette with return postage to:

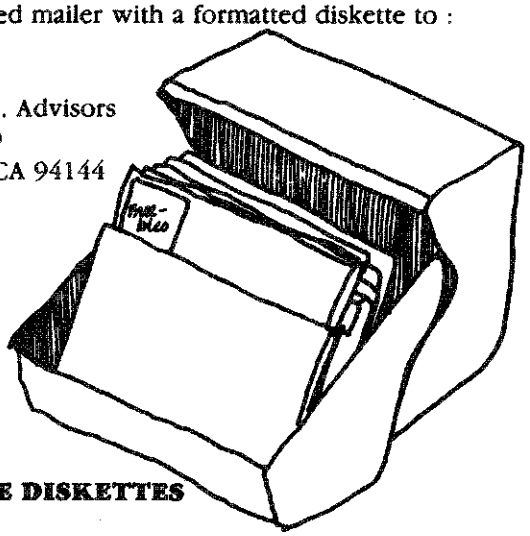
PC-TALK
Headland Press
P.O. Box 862
Tiburon, CA 94920

PC-FILE
If you would like to try a data base manager for no charge then send the unusual formatted diskette with return postage and mailer to:

PC-FILE
Jim Button
P.O. Box 5786
Bellevue, WA 98006

Free RatBAS
If you want an excellent tutorial on structured BASIC and a free translator to allow you to write more Pascal like BASIC code then send for a copy of the RatBAS translator program with documentation. Send a self-addresses stamped mailer with a formatted diskette to :

Brent Weaver
Wells Fargo Inv. Advisors
P.O. Box 44029
San Francisco, CA 94144



SOURCE CODE DISKETTES

DISK $1 $15

BASIC INPUT: Includes a professional level input routine that can be plugged into any BASIC program for simple, bug free input. You need only indicate the row, column, and word length. It also provides for prompt messages. The diskette also contains a sample structured program to use as an outline for easy to read code.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

REAL PROGRAMMERS:

Real Programmers do not eat Quiche.
Real Programmers start writing their programs without bothering with flowcharts because they intuitively grasp the complete system.
Real Programmers do not do maintenance.
Real Programmers write perfect code. Any bugs are a result of users changing the specifications.
Real Programmers sneer when you mention database managers.
Real Programmers tell a user the way they should have their system setup, after all only programmers fully understand a computer.
Real programmers never admit to having been a Computer Operator.

VIRTUAL IS ITS OWN REWARD.

IBM PC BULLETIN BOARDS:

If you have a modem and wish to connect with other PC'ers to exchange information, here is the current list:

(703) 560-0979 Annadale, Virginia SYSOP—
Wes Merchant (24 hrs, download, messages,bulletins.)

(301) 949-8848 Rockville, Maryland SYSOP—
Rich Schinnel (24 hrs, download, upload only. Password = IBM PC)

(301) 251-6273 Gaithersburg, Maryland SYSOP—
Larry Jordan (24 hrs, communications info. Password = IBM PC)

(703) 978-9592 Fairfax, Va SYSOP—
Don Withrow [Hostcomm] (24 hrs, download, upload, tips. Password = IBM PC)

(301) 460-0538 Silver Spring, MD SYSOP—
Ramona Landberg (24 hrs, upload, newsletter articles, Password = IBM PC)

DISK TOO SHALL PASS.

PC CONVENTIONS:

The Eighth West Coast Computer Faire will be held in San Francisco,CA March 18-20th. It will be held at the Civic Auditorium and Brooks Hall. IBM will be having a large exhibit.

There are 2 other San Francisco conventions expressly for the IBM PC this summer.The first is June 17-19th called the PC 83 Convention held in Brooks Hall, and another will be held in late August.

(703) 560-7803 Vienna, VA ABBS
with IBM PC Conference (24 hrs, download, upload, messages)

(301) 937-4339 Beltsville, MD—
Small People Software (24 hrs, games, messages.)

(404) 252-9438 Atlanta, GA SYSOP—
Rod Roark (24 hrs, download, upload, messages, tips, news)

(303) 773-2699 Denver, CO SYSOP—
Chris Van Essen (24 hrs, download, upload, Email, tips, buy)

(406) 656-9624 Billings, Montana SYSOP—
George Peck (9am-5pm, 11pm-7am Mountain time)

312) 259-8086 Chicago, IL SYSOP—
Gene Plantz (PCMODEM) (24 hrs, download, upload, messages)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WE KNOW YOU'RE SERIOUS ABOUT THE IBM PC.

SO ARE WE!

WE KNOW WHAT YOU WANT.

- *Detailed how-to articles*
- *Programming tips*
- *In-Depth product reviews*
- *Problem reporting and solutions*
- *Code listings*

WE'VE ALWAYS HAD IT ...

Some features in previous issues were a technical look at the system unit, number crunching, utility programs, asynchronous communications, monitor and printer character sets, personalizing DOS, BASIC vs. Pascal, menu generation programs, unleashing your printer, evaluation of hard disk drives, and plotting data with PC graphics.

WE ALWAYS WILL.

Features for 1983 will include extensive information about networks, database managers, spreadsheets, operating systems, the 8087 chip, professional program development, and much more.

WE KNOW WHICH MAGAZINE YOU NEED ...

PERSONAL COMPUTER AGE

The Definitive Journal for the IBM Personal Computer User

CALL TOLL FREE: **800-824-7888 operator 77**
California only: 800-852-7777 op. 77

I NEED WHAT YOU'VE GOT! **ENTER MY SUBSCRIPTION TODAY!**

☐ I WANT INFORMATION ABOUT BACK ISSUES, TOO!

☐ PAYMENT ENCLOSED ☐ BILL ME

CHARGE MY:

☐ VISA CARD NO. _____

☐ MASTERCARD EXPIRATION DATE _____

AUTHORIZED SIGNATURE _____

☐ 1 yr. (12 issues) $24.00
☐ 2 yrs. (24 issues) $45.00
☐ 3 yrs. (36 issues) $64.00
☐ NEW ☐ RENEWAL

FOREIGN SUBSCRIPTIONS
1 YR. (12 ISSUES)
CANADA/MEXICO $29.00

OTHER COUNTRIES,
WRITE FOR RATES
PAYMENTS MUST BE
IN U.S. FUNDS DRAWN
ON A U.S. BANK

NAME _____

ADDRESS _____

CITY _____

STATE _____ ZIP _____

Personal Computer Age

Department PJ4
P.O. Box 70725
Pasadena,
California 91107

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROGRAMMER'S JOURNAL

The Resource Newsletter for IBM PC Programmers

SUBSCRIPTION

Include your:

Name _____

Company _____

Address _____

City _____

State _____ Zip _____

1 year subscription rate:
$24 U.S. ($30 Canada, $40 other)
☐ Check enclosed
☐ Bill me

Signature _____
Begin my subscription with: ☐ First Issue ☐ Current Issue

Your system configuration:

Memory (K) 64 128 256+

Disk Drives 1 2 160 320 HD

Monitor: Color Mono

Languages you use: _____

Languages interested in: _____

Send to:
PROGRAMMER'S JOURNAL
2765 Potter St.
Eugene, OR 97405
Ph. (503) 687-0398

PROGRAMMER'S JOURNAL

The Resource Newsletter for IBM PC Programmers

SUBSCRIPTION

Include your:

Name _____

Company _____

Address _____

City _____

State _____ Zip _____

1 year subscription rate:
$24 U.S. ($30 Canada, $40 other)
☐ Check enclosed
☐ Bill me

Signature _____
Begin my subscription with: ☐ First Issue ☐ Current Issue

Your system configuration:

Memory (K) 64 128 256+

Disk Drives 1 2 160 320 HD

Monitor: Color Mono

Languages you use: _____

Languages interested in: _____

Send to:
PROGRAMMER'S JOURNAL
2765 Potter St.
Eugene, OR 97405
Ph. (503) 687-0398

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEXT TIME . . .

DOS TIPS
Expanding your knowledge of the Operating System

MENU PLANNING
Three deep is a thing of the past

MODULAR PROGRAMMING STYLE
Don't code yourself into a corner

TEXT EDITORS
A look at the programmer's Number 1 tool

PROGRAMMER'S QUIZ
A self-test for software authors

and more of the languages for the IBM PC.

$4.00

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED